Learning to combine primitive skills: A step towards versatile robotic manipulation[§]

Robin Strudel[*†], Alexander Pashevich[*‡], Igor Kalevatykh[†], Ivan Laptev[†], Josef Sivic[†], Cordelia Schmid[‡]

***Abstract*— Manipulation tasks such as preparing a meal or assembling furniture remain highly challenging for robotics and vision. Traditional task and motion planning (TAMP) methods can solve complex tasks but require full state observability and are not adapted to dynamic scene changes. Recent learning methods can operate directly on visual inputs but typically require many demonstrations and/or task-specific reward engineering. In this work we aim to overcome previous limitations and propose a reinforcement learning (RL) approach to task planning that learns to combine primitive skills. First, compared to previous learning methods, our approach requires neither intermediate rewards nor complete task demonstrations during training. Second, we demonstrate the versatility of our vision-based task planning in challenging settings with temporary occlusions and dynamic scene changes. Third, we propose an efficient training of basic skills from few synthetic demonstrations by exploring recent CNN architectures and data augmentation. Notably, while all of our policies are learned on visual inputs in simulated environments, we demonstrate the successful transfer and high success rates when applying such policies to manipulation tasks on a real UR5 robotic arm.**

I. Introduction

In this work we consider visually guided robotics manipulations and aim to learn robust visuomotor control policies for particular tasks. Autonomous manipulations such as assembling IKEA furniture [1] remain highly challenging given the complexity of real environments as well as partial and uncertain observations provided by the sensors. Successful methods for task and motion planning (TAMP) [2]–[4] achieve impressive results for complex tasks but often rely on limiting assumptions such as the full state observability and known 3D shape models for manipulated objects. Moreover, TAMP methods usually complete planning before execution and are not robust to dynamic scene changes.

Recent learning methods aim to learn visuomotor control policies directly from image inputs. Imitation learning (IL) [5]–[8] is a supervised approach that can be used to learn simple skills from expert demonstrations. One drawback of IL is its difficulty to handle new states that have not been observed during demonstrations. While increasing the number of demonstrations helps to alleviate this issue, an exhaustive sampling of action sequences and scenarios becomes impractical for long and complex tasks.

In contrast, reinforcement learning (RL) requires little supervision and achieves excellent results for some challenging tasks [9], [10]. RL explores previously unseen scenarios and, hence, can generalize beyond expert demonstrations. As full exploration is exponentially hard and becomes impractical for problems with long horizons, RL often relies on careful engineering of rewards designed for specific tasks.

Common tasks such as preparing food or assembling furniture require long sequences of steps composed of many different actions. Such tasks have long horizons and, hence, are difficult to solve by either RL or IL methods alone. To address this issue, we propose a RL-based method that learns to combine simple imitation-based policies. Our approach simplifies RL by reducing its exploration to sequences with a limited number of primitive actions, that we call *skills*.

Given a set of pre-trained skills such as "grasp a cube" or "pour from a cup", we train RL with sparse binary rewards corresponding to the correct/incorrect execution of the full task. While hierarchical policies have been proposed in the past [11], [12], our approach can learn *composite manipulations using no intermediate rewards and no demonstrations of full tasks*. Hence, the proposed method can be directly applied to learn new tasks. See Figure 1 for an overview of our approach.

Our skills are low-level visuomotor controllers learned from synthetic demonstrated trajectories with behavioral cloning (BC) [5]. Examples of skills include *go to the bowl*, *grasp the object*, *pour from the held object*, *release the held object*, etc. We automatically generate expert synthetic demonstrations and learn corresponding skills in simulated environments. We also minimize the number of required demonstrations by choosing appropriate CNN architectures and data augmentation methods. Our approach is shown to compare favorably to the state of the art [7] on the FetchPickPlace test environment [13]. Moreover, using recent techniques for domain adaptation [14] we demonstrate the successful transfer and high accuracy of our simulator-trained policies when tested on a real robot

We compare our approach with two classical methods: (a) an open-loop controller estimating object positions and applying a standard motion planner (b) a closed-loop controller adapting the control to re-estimated object positions. We show the robustness of our approach to a variety of perturbations. The perturbations include dynamic change of object positions, new object instances and temporary object occlusions. The versatility of learned policies comes from both the reactivity of the BC learned skills and the ability of the RL master policy to re-plan in case of failure. Our

[*]Equal contribution.

[†]Inria, École normale supérieure, CNRS, PSL Research University, 75005 Paris, France.

[‡]University Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.

[§]This work was funded in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

approach allows to compute adaptive control and planning in real-time.

In summary, this work makes the following contributions. (i) We propose to learn robust RL policies that combine BC skills to solve composite tasks. (ii) We present sample efficient training of BC skills and demonstrate an improvement compared to the state of the art. (iii) We demonstrate successful learning of relatively complex manipulation tasks with neither intermediate rewards nor full demonstrations. (iv) We successfully transfer and execute policies learned in simulation to real robot setups. (v) We show successful task completion in the presence of perturbations.

Our simulation environments and the code used in this work is publicly available on our website [15].

II. Related work

Our work is related to robotics manipulation such as grasping [16], opening doors [17], screwing the cap of a bottle [18] and cube stacking [19]. Such tasks have been addressed by various methods including imitation learning (IL) [20] and reinforcement learning (RL) [21].

Imitation learning (IL). A neural network is trained to solve a task by observing demonstrations. Approaches include behavioral cloning (BC) [5] and inverse reinforcement learning [8]. BC learns a function that maps states to expert actions [6], [7], whereas inverse reinforcement learning learns a reward function from demonstrations in order to solve the task with RL [19], [22], [23]. BC typically requires a large number of demonstrations and has issues with not observed trajectories. While these problems might be solved with additional expert supervision [6] or noise injection in expert demonstrations [24], we address them by improving the standard BC framework. We use recent state-of-the-art CNN architectures and data augmentation for expert trajectories. This permits to significantly reduce the number of required demonstrations and to improve performance.

Reinforcement learning (RL). RL learns to solve a tasks without demonstrations using exploration. Despite impressive results in several domains [9], [10], [17], [25], RL methods show limited capabilities when operating in complex and sparse-reward environments common in robotics. Moreover, RL methods typically require prohibitively large amounts of interactions with the environment during training. Hierarchical RL (HRL) methods alleviate some of these problems by learning a high-level policy modulating low-level workers. HRL approaches are generally based either on options [26] or a feudal framework [27]. The option methods learn a master policy that switches between separate skill policies [28]–[31]. The feudal approaches learn a master policy that modulates a low-level policy by a control signal [32]–[36]. Our approach is based on options but in contrast to the cited methods, we pretrain the skills with IL. This allows us to solve complex and sparse reward problems using significantly less interactions with the environment during training.

Combining RL and IL. A number of approaches combining RL and IL have been introduced recently. Gao et al. [37] use demonstrations to initialize the RL agent. In [38], [39] RL is used to improve expert demonstrations, but does not learn hierarchical policies. Demonstrations have also been used to define RL objective functions [40], [41] and rewards [42]. Das et al. [11] combine IL and RL to learn a hierarchical policy. Unlike our method, however, [11] requires full task demonstrations and task-specific reward engineering. Moreover, the addressed navigation problem in [11] has a much lower time horizon compared to our tasks. [11] also relies on pre-trained CNN representations which limits its application domain. Le at al. [12] train low-level skills with RL, while using demonstrations to switch between skills. In a reverse manner, we use IL to learn low-level control and then deploy RL to find appropriate sequences of pre-trained skills. The advantage is that our method can learn a variety of complex manipulations without full task demonstrations. Moreover, [11], [12] learn discrete actions and cannot be directly applied to robotics manipulations that require continous control.

In summary, none of the methods [11], [12], [38], [39] is directly suitable for learning complex robotic manipulations due to requirements of dense rewards [11], [39] and state inputs [38], [39], limitations to short horizons and discrete actions [11], [12], the requirement of full task demonstrations [11], [12], [38], [39] and the lack of learning of visual representations [11], [38], [39]. Moreover, our skills learned from synthetic demonstrated trajectories outperform RL based methods, see Section V-D.

III. Approach

Our RLBC approach aims to learn multi-step policies by combining reinforcement learning (RL) and pre-trained skills obtained with behavioral cloning (BC). We present BC and RLBC in Sections III-A and III-B. Implementation details are given in Section III-C.

A. Skill learning with behavioral cloning

Our first goal is to learn basic skills that can be composed into more complex policies. Given observation-action pairs $\mathcal{D} = \{(o_t, a_t)\}$ along expert trajectories, we follow the behavioral cloning approach [5] and learn a function approximating the conditional distribution of the expert policy $\pi_{\mathrm{E}}(a_t|o_t)$ controlling a robot arm. Our observations $o_t \in \mathcal{O} = \mathbb{R}^{\mathrm{H}\times\mathrm{W}\times\mathrm{M}}$ are sequences of the last M depth frames. Actions $a_t = (\mathbf{v}_t, \boldsymbol{\omega}_t, g_t)$, $a_t \in \mathcal{A}^{\mathrm{BC}}$ are defined by the end-effector linear velocity $\mathbf{v}_t \in \mathbb{R}^3$ and angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$ as well as the gripper openness state $g_t \in \{0, 1\}$.

We learn the deterministic skill policies $\pi_s : \mathcal{O} \rightarrow \mathcal{A}^{\mathrm{BC}}$ approximating the expert policy π_{E}. Given observations o_t with corresponding expert (ground truth) actions $a_t = (\mathbf{v}_t, \boldsymbol{\omega}_t, g_t)$, we represent π_s with a convolutional neural network (CNN) and learn network parameters (θ, η) such that predicted actions $\pi_s(o_t) = (\hat{\mathbf{v}}_t, \hat{\boldsymbol{\omega}}_t, \hat{g}_t)$ minimize the loss $L_{\mathrm{BC}}(\pi_s(o_t), a_t) = \lambda \left\| [\hat{\mathbf{v}}_\mathbf{t}, \hat{\boldsymbol{\omega}}_t] - [\mathbf{v}_t, \boldsymbol{\omega}_t] \right\|_2^2 + (1 - \lambda) \left(g_t \log \hat{g}_t + (1 - g_t) \log (1 - \hat{g}_t) \right)$, where $\lambda \in [0, 1]$ is a scaling factor which we empirically set to 0.9.

Our network architecture is presented in Figure 1(right). When training a skill policy π_s^i, such as reaching, grasping or pouring, we condition the network on the skill using the recent FiLM architecture [43]. Given the one-hot encoding s^i of a skill i, we use s^i as input to the FiLM generator which performs affine transformations of the network feature maps. FiLM conditions the network on performing a given skill,

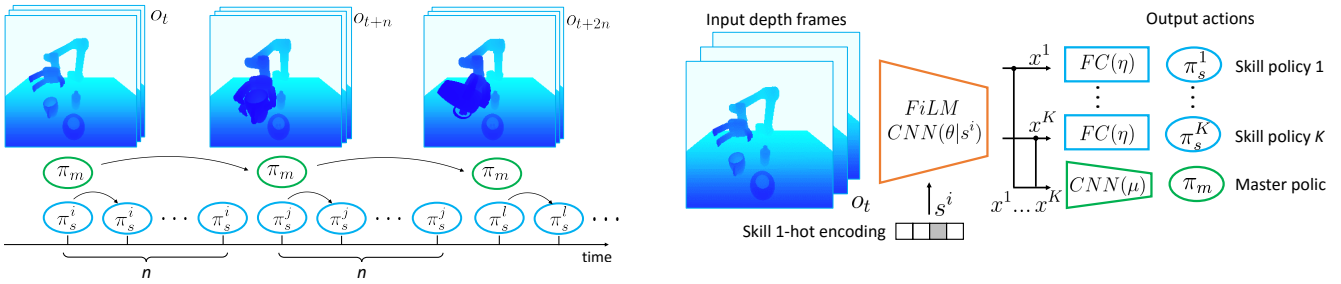


Fig. 1: Illustration of our approach. (Left): Temporal hierarchy of master and skill policies. The master policy π_m is executed at a coarse interval of n time-steps to select among K skill policies $\pi_s^1 \dots \pi_s^K$. Each skill policy generates control for a primitive action such as *grasping* or *pouring*. (Right): CNN architecture used for the skill and master policies.

which permits learning a shared representation for all skills. Given an observation o_t, the network $CNN(\theta|s^i)$ generates a feature map x_t^i conditioned on skill i. The spatially-averaged x_i is linearly mapped with $FC(\eta)$ to the action of π_s^i.

B. RLBC approach

We wish to solve composite manipulations without full expert demonstrations and with a single sparse reward. For this purpose we rely on a high-level *master policy* π_m controlling the pre-trained *skill policies* π_s at a coarse timescale. To learn π_m, we follow the standard formulation of reinforcement learning and maximize the expected return $\mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ given rewards r_t. Our reward function is sparse and returns 1 upon successful termination of the task and 0 otherwise. The RL master policy $\pi_m : \mathcal{O} \times \mathcal{A}^{\text{RL}} \rightarrow [0, 1]$ chooses one of the K skill policies to execute the low-level control, i.e., the action space of π_m is discrete: $\mathcal{A}^{\text{RL}} = \{1, \dots, K\}$. Note, that our sparse reward function makes the learning of deep visual representations challenging. We, therefore, train π_m using visual features x_t^i obtained from the BC pre-trained $CNN(\theta|s^i)$. Given an observation o_t, we use the concatenation of skill-conditioned features $\{x_t^1, \dots, x_t^K\}$ as input for the master $CNN(\mu)$, see Figure 1(right).

To solve composite tasks with sparse rewards, we use a coarse timescale for the master policy. The selected skill policy controls the robot for n consecutive time-steps before the master policy is activated again to choose a new skill. This allows the master to focus on high-level task planning rather than low-level motion planning achieved by the skills. We expect the master policy to recover from unexpected events, for example, if an object slips out of a gripper, by re-activating an appropriate skill policy. Our combination of the master and skill policies is illustrated in Figure 1(left).

RLBC algorithm. The pseudo-code for the proposed approach is shown in Algorithm 1. The algorithm can be divided into three main steps. First, we collect a dataset of expert trajectories $\mathcal{D}_k$ for each skill policy π_s^k. For each policy, we use an expert script that has an access to the full state of the environment. Next, we train a set of skill policies $\{\pi_s^1, \dots, \pi_s^K\}$. We sample a batch of state-action pairs and update parameters of convolutional layers θ and the skills linear layer parameters η. Finally, we learn the master π_m using the pretrained skill policies and the frozen parameters θ. We collect episode rollouts by first choosing a skill policy with the master and then applying the selected skill to the environment for n time-steps. We update the master policy weights μ to maximize the expected sum of rewards.

Algorithm 1 RLBC

*** Collect expert data ***
for $k \in \{1, \dots, K\}$ **do**
    Collect an expert dataset $\mathcal{D}_k$ for the skill policy π_s^k
*** Train $\{\pi_s^1, \dots, \pi_s^K\}$ by solving: ***
$\theta, \eta = \arg\min_{\theta,\eta} \sum_{k=1}^{K} \sum_{(o_t,a_t)\in\mathcal{D}_k} L_{\text{BC}}(\pi_s^k(o_t), a_t)$
while task is not solved **do**
    *** Collect data for the master policy ***
    $\mathcal{E} = \{\}$ ▷ Empty storage for rollouts
    for episode_id $\in \{1, \dots,$ ppo_num_episodes$\}$ **do**
        $o_0 =$ new_episode_observation()
        $t = 0$
        while episode is not terminated **do**
            $k_t \sim \pi_m(o_t)$ ▷ Choose the skill policy
            $o_{t+n}, r_{t+n} =$ perform_skill$(\pi_s^{k_t}, o_t)$
            $t = t + n$
        $\mathcal{E} = \mathcal{E} \cup \{(o_0, k_0, r_n, o_n, k_n, r_{2n}, o_{2n}, \dots)\}$
    *** Make a PPO step for the master policy on $\mathcal{E}$ ***
    $\mu =$ ppo_update$(\pi_m, \mathcal{E})$

C. Approach details

Skill learning with BC. We use ResNet-18 for the $CNN(\theta|s^i)$, which we compare to VGG16 and ResNet-101 in Section V-A. We augment input depth frames with random translations, rotations and crops. We also perform viewpoint augmentation and sample the camera positions on a section of a sphere centered on the robot and with a radius of 1.40 m. We uniformly sample the yaw angle in $[-15°, 15°]$, the pitch angle in $[15°, 30°]$, and the distance to the robot base in $[1.35m, 1.50m]$. The impact of both augmentations is evaluated in Section V-B. We normalize the ground truth of the expert actions to have zero mean and a unit variance and normalize the depth values of input frames to $[-1, 1]$. We learn BC skills using Adam [44] with the learning rate 10^{-3} and a batch size 64. We also use Batch Normalization [45].

Task learning with RL. We learn the master policies with the PPO [46] algorithm using the open-source implementation [47] where we set the entropy coefficient to 0.05, the value loss coefficient to 1, and use 8 episode rollouts for the PPO update. For the RLBC method, the concatenated skill features $\{x_t^1, \dots, x_t^K\}$ are processed with the master network $CNN(\mu)$ having 2 convolutional layers with 64 filters of size 3×3. During pre-training of skill policies we update the parameters (θ, η). When training the master policy, we only update μ while keeping (θ, η) parameters fixed. We train RLBC using 8 different random seeds in parallel and evaluate the best one.

Real robot transfer. To deploy our method on the real

robot, we use a state-of-the-art technique of learning sim2real transfer based on data augmentation with domain randomization [14]. This method uses a proxy task of cube position prediction and a set of basic image transformations to learn a sim2real data augmentation function for depth images. We augment the depth frames from synthetic expert demonstrations with this method and, then, train skill policies. Once the skill policy is trained on these augmented simulation images, it is directly used on the real robot.

IV. Experimental setup

This section describes the setup used to evaluate our approach. First, we present the robot environment and the different tasks in Sections IV-A and IV-B. Next, we describe the synthetic dataset generation and skill definition for each task in Sections IV-C and IV-D.

A. Robot and agent environment

For our experiments we use a 6-DoF UR5 robotic arm with a 3 finger Robotiq gripper, see Figure 2. In simulation, we model the robot with the `pybullet` physics simulator [48]. For observation, we record depth images with the Microsoft Kinect 2 placed in front of the arm. The agent takes as input the three last depth frames $o_t \in \mathbb{R}^{224\times224\times3}$ and commands the robot with an action $a_t \in \mathbb{R}^7$. The control is performed at 10 Hz frequency.

B. UR5 tasks

For evaluation, we consider 3 tasks: UR5-Pick, UR5-Bowl and UR5-Breakfast. The **UR5-Pick** task picks up a cube of a size between 3.5 cm and 8.0 cm and lifts it up, see Figure 2a. In **UR5-Bowl** the robot has to grasp the cube and place it in the bowl, see Figure 2b. The **UR5-Breakfast** task contains a cup, a bottle and a bowl as shown in Figure 2c. We use distinct ShapeNet [49] object instances for the training and test sets (27 bottles, 17 cups, 32 bowls in each set). The robot needs to pour ingredients from the cup and the bottle in the bowl. In all tasks, the reward is positive if and only if the task goal is reached. The maximum episode lengths are 200, 600, and 2000 time-steps for UR5-Pick, UR5-Bowl, and UR5-Breakfast correspondingly.

C. Synthetic datasets

We use the simulated environments to create a synthetic training and test set. For all our experiments, we collect trajectories with random initial configurations where the objects and the end-effector are allocated within a workspace of $80 \times 40 \times 20\,\text{cm}^3$. The synthetic demonstrations are collected using an expert script designed for each skill. The script has access to the full state of the system including the states of the robot and the objects. To generate synthetic demonstrations, we program end-effector trajectories and use inverse kinematics (IK) to generate corresponding trajectories in the robot joints space. Each demonstration consists of multiple pairs of the three last camera observations and the robot control command performed by the expert script. For UR5-Pick, we collect 1000 synthetic demonstrated trajectories for training. For UR5-Bowl and UR5-Breakfast, we collect a training dataset of 250 synthetic demonstrations. For evaluation of each task, we use 100 different initial configurations in simulation and 20 trials on the real robot.

Demos	VGG16-BN	ResNet-18	ResNet-101
20	1%	1%	0%
50	9%	5%	5%
100	37%	65%	86%
1000	95%	**100%**	**100%**

TABLE I: Evaluation of BC skills trained with different CNN architectures and number of demonstrations on the UR5-Pick task in simulation.

Demos	None	Standard	Viewpoint	Standard & Viewpoint
20	1%	49%	39%	75%
50	5%	81%	79%	93%
100	65%	97%	**100%**	**100%**

TABLE II: Evaluation of ResNet-18 BC skills trained with different data augmentations on UR5-Pick task in simulation.

D. Skill definition

UR5-Pick task is defined as a single skill. For UR5-Bowl and UR5-Breakfast, we consider a set of skills defined by expert scripts. For UR5-Bowl, we define four skills: (a) go to the cube, (b) go down and grasp, (c) go up, and (d) go to the bowl and open the gripper. For UR5-Breakfast, we define four skills: (a) go to the bottle, (b) go to the cup, (c) grasp an object and pour it to the bowl, and (d) release the held object. We emphasize that the expert dataset does not contain full task demonstrations and that all our training is done in simulation. When training the RL master, we execute selected skills for 60 consecutive time-steps for the UR5-Bowl task and 220 time-steps for the UR5-Breakfast task.

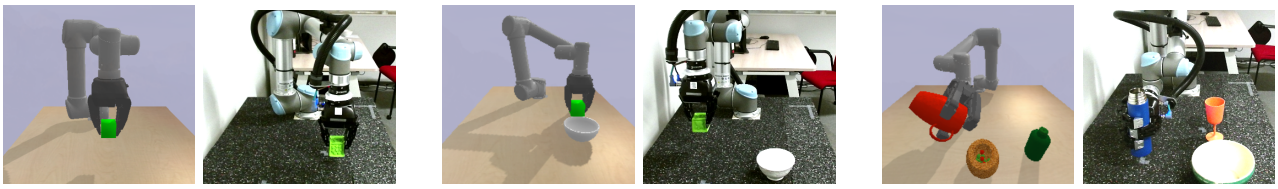
V. Evaluation of BC skill learning

This section evaluates the different parameters of the BC skill training for the UR5-Pick task and a comparison with the state of the art. First, we evaluate the impact of the CNN architecture and data augmentation on the skill performance in Sections V-A and V-B. Then, we show that the learned policies transfer to a real-robot in Section V-C. Finally, we compare the BC skills with the state of the art in Section V-D.

A. CNN architecture for BC skill learning

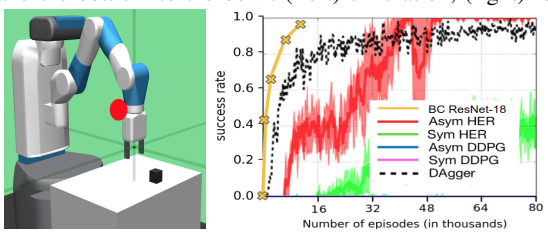
Given the simulated UR5-Pick task illustrated in Figure 2a(left), we compare BC skill networks trained with different CNN architectures and varying number of expert demonstrations. Table I compares the success rates of policies with VGG and ResNet architectures. Policies based on the VGG architecture [50] obtain success rate below 40% with 100 training demonstrations and reach 95% with 1000 demonstrations. ResNet [51] based policies have a success rate above 60% when trained on a dataset of 100 demonstrations and reach 100% with 1000 demonstrations. Overall ResNet-101 has the best performance closely followed by ResNet-18 and outperforms VGG significantly. To conclude, we find that the network architecture has a fundamental impact on the BC performance. In the following experiments we use ResNet-18 as it presents a good trade-off between performance and training time.

When examining why VGG-based BC has a lower success rate, we observe that it has higher validation errors compared to ResNet. This indicates that VGG performs worse on the



(a) UR5-Pick (b) UR5-Bowl (c) UR5-Breakfast

Fig. 2: UR5 tasks used for evaluation: (a) task of picking up the cube, (b) task of bringing the cube to the bowl, (c) task of pouring the cup and the bottle into the bowl. (Left) simulation, (right) real robot.



(a) FetchPickPlace (b) Comparison with [7]

Fig. 3: Comparison of BC ResNet-18 with state of the art [7] on the FetchPickPlace task. BC ResNet-18 results are reported for 200 different initial configurations.

level of individual steps and is hence expected to result in higher compounding errors.

B. Evaluation of data augmentation

We evaluate the impact of different types of data augmentations in Table II. We compare training without data augmentation with 3 variants: (1) random translations, rotations and crops, as is standard for object detection, (2) record each expert synthetic demonstration from 10 varying viewpoints and (3) the combination of (1) and (2).

Success rates for UR5-Pick on datasets with 20, 50 and 100 demonstrations are reported in Table II. We observe that data augmentation is particularly important when only a few demonstrations are available. For 20 demonstrations, the policy trained with no augmentation performs at 1% while the policy trained with standard and viewpoint augmentations together performs at 75%. The policy trained with a combination of both augmentation types performs the best and achieves 93% and 100% success rate for 50 and 100 demonstrations respectively. In summary, data augmentation allows a significant reduction in the number of expert trajectories required to solve the task.

C. Real robot experiments

We evaluate our method on the real-world UR5-Pick illustrated in Figure 2a(right). We collect demonstrated trajectories in simulation and train the BC skills network applying standard, viewpoint and sim2real augmentations. We show that our approach transfers well to the real robot using no real images. The learned policy manages to pick up cubes of 3 different sizes correctly in 20 out of 20 trials.

D. Comparison with state-of-the-art methods

One of the few test-beds for robotic manipulation is FetchPickPlace from OpenAI Gym [13] implemented in `mujoco` [52], see Figure 3a. The goal for the agent is to pick up the cube and to move it to the red target (see Figure 3a). The agent observes the three last RGB-D images from a camera placed in front of the robot $o_t \in \mathbb{R}^{100\times100\times4\times3}$. The positions of the cube and the target are set at random for each trial. The reward of the task is a single sparse reward of success. The maximum length of the task is 50 time-steps.

For a fair comparison with [7], we do not use any data augmentation. We report the success rate of ResNet-18 policy in Figure 3b. We follow [7] and plot the success rate of both RL and IL methods with respect to the number of episodes used (either trial episodes or demonstrations). Our approach outperforms the policies trained with an imitation learning method DAgger [6] in terms of performance and RL methods such as HER [53] and DDPG [54] in terms of data-efficiency. According to [7], DAgger does not reach 100% even after $8{*}10^4$ demonstrations despite the fact that it requires an expert during training. HER reaches the success rate of 100% but requires about $4 * 10^4$ trial episodes. Our approach achieves the 96% success rate using 10^4 demonstrations.

Our policies differ from [7] mainly in the CNN architecture. Pinto et al. [7] use a simple CNN with 4 convolutional layers while we use ResNet-18. Results of this section confirm the large impact of the CNN architecture on the performance of visual BC policies, as was already observed in Table I.

VI. Evaluation of RLBC

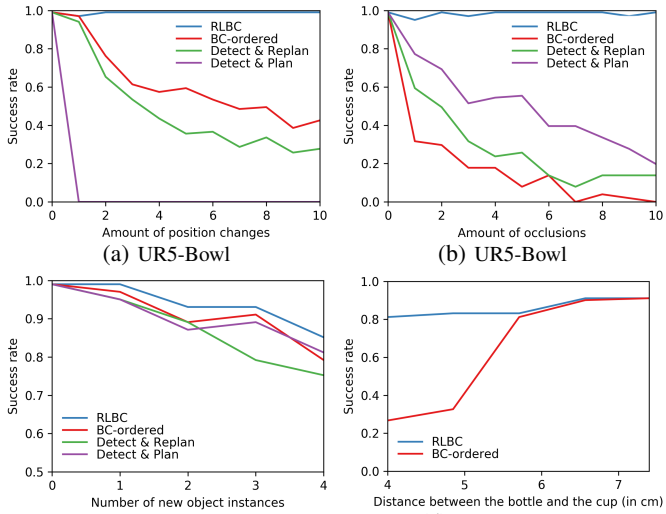
This section evaluates the proposed RLBC approach and compares it to baselines introduced in Section VI-A. First, we evaluate our method on UR5-Bowl in Section IV-B. We then test the robustness of our approach to various perturbations such as dynamic changes of object positions, dynamic occlusions, unseen object instances and the increased probability of collisions due to small distances between objects. We show that RLBC outperforms the baselines on those scenarios both in simulation and on a real robot. Note, that our real robot experiments are performed with skills and master policies that have been trained exclusively in simulation using sim2real augmentation [14]. We use the same policies for all perturbation scenarios. Qualitative results of our method are available in the **supplementary video**.

A. Baseline methods

We compare RLBC with 3 baselines: (a) a fixed sequence of BC skills following the manually pre-defined correct order (BC-ordered); (b) an open-loop controller estimating positions of objects and executing an expert script (Detect & Plan); (c) a closed-loop controller performing the same estimation-based control and replanning in case if object positions change substantially (Detect & Replan). We use the same set of skills for RLBC and BC-ordered. We train the position estimation network using a dataset of 20.000 synthetic depth images with randomized object positions. All networks use ResNet-18 architecture and are trained with the standard, viewpoint and sim2real augmentations described in Section V-B.

UR5-Bowl perturbations	Detect & Plan	Detect & Replan	BC-ordered	RLBC
No perturbations	17/20	16/20	17/20	**20/20**
Moving objects	0/20	12/20	13/20	**20/20**
Occlusions	17/20	10/20	2/20	**18/20**
New objects	16/20	14/20	15/20	**18/20**

TABLE III: Comparison of RLBC with 3 baselines on the real-world UR5-Bowl task with dynamic changes of the cube position, dynamic occlusions and new object instances.



(a) UR5-Bowl (b) UR5-Bowl (c) UR5-Bowl (d) UR5-Breakfast

Fig. 4: Performance of RLBC and baseline methods in simulated environments under perturbations: (a) Dynamic changes of cube position; (b) Dynamic occlusions; (c) Replacing the cube by unseen objects; (d) Decreasing the distance between objects.

B. Results on UR5-Bowl with no perturbations

We first evaluate RLBC and the three baselines on the UR5-Bowl task (see Figure 2b). When tested in simulation, all the baselines and RLBC manage to perfectly solve the task. On the real-world UR5-Bowl task, BC-ordered and Detect & Plan baselines sometimes fail to grasp the object which leads to task failures (see Table III, first row). On the contrary, RLBC solves the task in all 20 episodes given its ability to re-plan the task in the cases of failed skills.

We have also attempted to solve the simulated UR5-Bowl task without skills by learning an RL policy performing low-level control. We have used ImageNet pre-trained ResNet-18 to generate visual features. The features were then used to train low-level RL control policy with PPO. Whereas such a low-level RL policy did not solve the task a single time after 10^4 episodes, RLBC reaches 100% after 400 episodes.

C. Robustness to perturbations

Robustness to dynamic changes in object position. We evaluate RLBC against the baselines in the UR5-Bowl scenario where the cube is moved several times during the episode. We plot success rates evaluated in simulation with respect to the number of position changes in Figure 4a. We observe the stability of RLBC and the fast degradation of all baselines. As both RLBC and BC-ordered use the same set of skills, the stability of RLBC comes from the learned skill combination. The "Moving objects" row in Table III reports results for 3 moves of the cube evaluated on the real robot. Similarly to the simulated results, we observe excellent results of RLBC and the degraded performance for all the baselines.

Robustness to occlusions. We evaluate the success of UR5-Bowl task under occlusions. Each occlusion lasts 3 seconds and covers a large random part of the workspace by a cardboard. Figure 4b shows success rates with respect to the number of occlusions in the simulated UR5-Bowl environment. Similarly to the perturbation results in Figure 4a, RLBC demonstrates high robustness to occlusions while the performance of other methods quickly degrades. The "Occlusions" row in Table III reports results for a single occlusion performed during the real-robot evaluation. Baseline methods are strongly influenced by occlusions except Detect & Plan which performs well unless occlusion happens during the first frames. Our RLBC policy performs best compared to other methods.

Robustness to new object instances. We evaluate the robustness of methods to the substitution of a cube by other objects not seen during the training of UR5-Bowl task. Figure 4c shows the success rate of RLBC and other methods with respect to the number of new objects in simulation. The novel objects are ordered by their dissimilarity with the cube. The difficulty of grasping unseen objects degrades the performance of grasping skills. In contrast to other methods RLBC is able to automatically recover from errors by making several grasping attempts. Table III reports corresponding results on a real robot where the cube has been replaced by 10 unseen objects. Similarly to the other perturbations we observe superior performance of RLBC.

Impact of the distance between objects. We vary the distance between a bottle and a cup in the UR5-Breakfast task. The smaller distance between objects A and B implies higher probability of collision between a robot and A when grasping B behind A. The choice of the grasping order becomes important in such situations. While our method is able to learn the appropriate grasping order to maximize the chance of completing the task, the BC-ordered and other baselines use pre-defined order. Figure 4d demonstrates the performance of RLBC and BC-ordered for different object distances in the simulated UR5-Breakfast task. As expected, RLBC learns the correct grasping order and avoids most of collisions. The performance of BC-ordered strongly degrades with the decreasing distance. In the real-world evaluation, both RLBC and ordered skills succeed in 16 out of 20 episodes when the distance between objects is larger than 10 cm. However, the performance of BC-ordered drops to 8/20 when the cup and the bottle are at 4cm from each other. In contrast, RLBC chooses the appropriate object to avoid collisions and succeeds in 16 out of 20 trials.

VII. Conclusion

This paper presents a method to learn combinations of primitive skills. Our method requires no full-task demonstrations nor intermediate rewards and shows excellent results in simulation and on a real robot. We demonstrate the versatility of our approach in challenging settings with dynamic scene changes. Future work will include learning multiple tasks with shared skills and addressing contact-rich tasks.

REFERENCES

[1] F. Suarez-Ruiz, X. Zhou, and Q.-C. Pham, "Can robots assemble an IKEA chair?" *Science Robotics*, 2018. 1

[2] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *ICRA*, 2014. 1

[3] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *ICIRS*, 2014. 1

[4] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *IJCAI*, 2015. 1

[5] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *NIPS*, 1989. 1, 2

[6] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv*, 2014. 1, 2, 5

[7] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric Actor Critic for Image-Based Robot Learning," in *RSS*, 2018. 1, 2, 5

[8] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000. 1, 2

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015. 1, 2

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016. 1, 2

[11] A. Das, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, "Neural modular control for embodied question answering," in *CoRL*, 2018. 1, 2

[12] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. D. III, "Hierarchical imitation and reinforcement learning," in *ICML*, 2018. 1, 2

[13] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *arXiv*, 2018. 1, 5

[14] A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid, "Learning to augment synthetic images for sim2real policy transfer," in *IROS*, 2019. 1, 4, 5

[15] "Learning to combine primitive skills: A step towards versatile robotic manipulation, project website," *https://www.di.ens.fr/willow/research/rlbc/*. 2

[16] T. Lampe and M. Riedmiller, "Acquiring visual servoing reaching and grasping skills using neural reinforcement learning," in *IJCNN*, 2013. 2

[17] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation," in *ICML*, 2016. 2

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2015 2

[19] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation," *arXiv*, 2017 2

[20] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *NIPS*, 2017. 2

[21] M. A. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. V. de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing - Solving sparse reward tasks from scratch," in *MLR*, 2018 2

[22] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *NIPS*, 2016. 2

[23] V. Kumar, A. Gupta, E. Todorov, and S. Levine, "Learning dexterous manipulation policies from experience and imitation," *arXiv*, 2016 2

[24] M. Laskey, J. Lee, R. Fox, A. D. Dragan, and K. Goldberg, "DART: Noise injection for robust imitation learning," in *CoRL*, 2017 2

[25] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: a survey," *IJRR*, vol. 32, no. 11, 2013. 2

[26] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999. 2

[27] P. Dayan and G. Hinton, "Feudal Reinforcement Learning," in *NIPS*, 1993. 2

[28] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *ICLR*, 2018 2

[29] Y. Lee, S.-H. Sun, S. Somasundaram, E. S. Hu, and J. J. Lim, "Composing complex skills by learning transition policies with proximity reward induction," in *ICLR*, 2019. 2

[30] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture." in *AAAI*, 2017. 2

[31] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," in *ICLR*, 2017 2

[32] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine, "Latent space policies for hierarchical reinforcement learning," in *ICML*, 2018 2

[33] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *NIPS*, 2018 2

[34] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *ICML*, 2017 2

[35] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *NIPS*, 2016. 2

[36] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, "Learning an embedding space for transferable robot skills," in *ICLR*, 2018 2

[37] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," in *ICLR workshop*, 2018 2

[38] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," in *UAI*, 2018. 2

[39] W. Sun, J. A. Bagnell, and B. Boots, "Truncated horizon policy search: Combining reinforcement learning & imitation learning," in *ICLR*, 2018. 2

[40] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep Q-Learning from demonstrations," in *AAAI*, 2018. 2

[41] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *ICRA*, 2018 2

[42] Y. Zhu, Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv*, 2018 2

[43] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville, "Film: Visual reasoning with a general conditioning layer," in *AAAI*, 2018. 2

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015. 3

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015 3

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, 2017 3

[47] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," *GitHub*, 2018. 3

[48] E. Courmans and Y. Bai, "PyBullet, Python module for physics simulation, robotics and machine learning," *GitHub*, 2016. 4

[49] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An information-rich 3D model repository," *arXiv*, 2015 4

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2014. 4

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016. 4

[52] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IROS*, 2012 5

[53] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NIPS*, 2017. 5

[54] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2016 5