Learning Object Manipulation Skills via Approximate State Estimation from Real Videos

Vladimír Petrík[1*], **Makarand Tapaswi**[2*], **Ivan Laptev**[2], **Josef Šivic**[1]
[1]CIIRC, Czech Technical University in Prague [2]Inria
{vladimir.petrik,josef.sivic}@cvut.cz, {makarand.tapaswi,ivan.laptev}@inria.fr

Abstract: Humans are adept at learning new tasks by watching a few instructional videos. On the other hand, robots that learn new actions either require a lot of effort through trial and error, or use expert demonstrations that are challenging to obtain. In this paper, we explore a method that facilitates learning object manipulation skills directly from videos. Leveraging recent advances in 2D visual recognition and differentiable rendering, we develop an optimization based method to estimate a coarse 3D state representation for the hand and the manipulated object(s) without requiring any supervision. We use these trajectories as dense rewards for an agent that learns to mimic them through reinforcement learning. We evaluate our method on simple single- and two-object actions from the Something-Something dataset. Our approach allows an agent to learn actions from single videos, while watching multiple demonstrations makes the policy more robust. We show that policies learned in a simulated environment can be easily transferred to a real robot.

Keywords: Learning from videos, coarse 3D state estimation, imitation learning

1 Introduction

Imagine that you want to assemble your bicycle. Humans are able to watch and learn from other people that demonstrate how to perform these actions, *e.g.*, by watching videos from YouTube [1]. We are interested in providing intelligent agents the ability to learn object manipulation skills by watching videos (see Fig. 1).

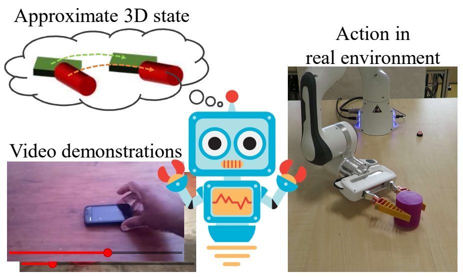

Figure 1: Our robot watches a few video demonstrations and learns to perform the observed action by estimating a coarse hand-object 3D state.

Learning from demonstrations [2] aims at teaching robots to perform various actions based on demonstrations, often performed by *experts*. While previous works in this area have required humans to teleoperate the robot [3], we wish to address a setting where the robot learns directly from video demonstrations. In particular, we are interested in understanding how to bridge the gap between the observed moving pixels (videos) and instructions that a robot can understand and execute. We address this question in two steps (see Fig. 2). First, building upon advances in computer vision and differentiable rendering, we propose *Real2Sim*, a method that lifts real world 2D videos to an approximate 3D state space representation of the hand and the manipulated objects (Sec. 3). Second, we use these automatically extracted trajectories along with reinforcement learning (RL) to learn policies that execute the actions in a 3D simulation environment corresponding to the real robot set-up and on a real robot (Sec. 4).

While there have been recent works on estimating detailed 3D meshes for hands and manipulated objects [4], these methods do not generalize well to out-of-domain videos. Instead, we pursue an alternative solution: we extract only a coarse representation of the scene where we approximate the hand as a cylinder and objects as cuboids. Leveraging 2D hand-object detectors [5] and pixel-segmentation methods [6], we are able to reconstruct a coarse 3D state representation that shares

*indicates equal contribution

perceptual similarities with the video. Even though the estimated states lack details (*e.g*. grasping an object), the coarse states of the dynamic scene act as a guide (a dense reward) for RL in a simulated environment, that in turn resembles the real world sufficiently allowing to transfer the learned policy to the real world. Our intermediate state representation allows a tangible and interpretable parsing of the video, and is specially effective at modeling actions that include object motion.

We investigate the following three questions in this work. First, we are interested in studying whether we can learn to perform simple object manipulations based on a single demonstration. We evaluate our approach on 9 simple object manipulation actions from the Something-Something action recognition dataset [7] and show that some *clean* videos are indeed capable of teaching an agent how to perform the task. Second, we hypothesize and empirically show that a robust policy can be trained by watching a handful of videos (a few-shot setting), thus reducing the impact of variable quality of individual demonstrations (Sec. 5). Finally, we are interested in studying how the object size or the initial gripper position impacts performance. We show that automatic domain randomization [8] presents a form of curriculum learning strategy that allows the agent to learn the task with gradually increasing complexity. We propose a challenging benchmark over the 9 actions, with randomized object sizes and initial gripper positions. Our action-specific metrics analyze whether the robot is able to perform the action correctly. We also demonstrate how the learned policies can be transferred to a real robot. We will make the code and data publicly available at `https://git.io/JTPkj`.

2 Related Work

We discuss recent advances in extracting 3D state/mesh representations from 2D images or videos and learning from video demonstrations.

Differentiable rendering. Understanding the 3D world in a projected image is a classical computer vision problem [9, 10]. However, obtaining ground-truth labels for 3D meshes is much harder than 2D object bounding boxes leading to challenges in 3D modeling [11]. One approach is to learn a *de-renderer* [12], an encoder that attempts to predict a structured scene representation, which can reconstruct the original scene through a graphics engine. This is extended to study object dynamics through visual de-animation [13]. Alternatively, there have been attempts to reconstruct 3D object meshes by incorporating graph neural networks [14] or intermediate voxel spaces [11]. Recent developments in differentiable neural rendering [15, 16], have enabled approaches that estimate both the 3D texture and shape [17] while only requiring 2D supervision (*e.g*. a segmentation mask). Ours is a parameter-free method that can estimate an approximate 3D representation from a single video.

Hands and objects in 3D. Joint understanding of hands and objects has implications from action recognition [18, 19] to virtual or agumented reality [20, 21]. Inspired by the SMPL person model [22], Romero et al. [23] develop MANO, an articulated hand model has helped researchers focus on studying various hand deformations [24]. Recently, there have been large data collection efforts for the joint study of hands and objects: as 2D bounding boxes [5, 25], or 3D pose estimates [26]. Nevertheless, joint 3D reconstruction of hand and object, especially during manipulation remains a challenging problem. There have been some efforts in this direction, such as leveraging synthetic datasets and utilizing manipulation constraints [4], or addressing the challenges in 3D annotation through temporal consistency in a video [27]. In this work, we propose an alternative solution. We hypothesize that a coarse 3D state representation is sufficient to teach a robot to perform these actions, and propose an optimization procedure to estimate hand-object trajectories from monocular videos.

Learning from demonstrations. Enabling a robot to learn policies based on expert demonstrations is a well-studied problem [2, 28, 29]. Learning from demonstrations (LfD) circumvents designing task-specific reward functions [30] that are often challenging to devise. Many works in this direction leverage humans teleoperating a robot, *e.g*. [3, 31, 32]. In contrast, we are interested in leveraging the large diversity of videos where people perform various object manipulations as our demonstrations.

Learning from videos include directions such as estimating perceptual reward functions from a small number of demonstrations [33]; learning a visual representation via multiple views and metric learning, followed by learning a policy using a single third person demonstration [34]; imitation from observation by learning to predict different viewpoints [35]; or very recently, learning pixel-level translation to convert human demonstration to the robot's perspective [36]. Our goal is similar, but our approach differs. Without the need for multiple views, we estimate a 3D physical state representation through an optimization algorithm, thus mapping the video to a simulated environment. Parallel to

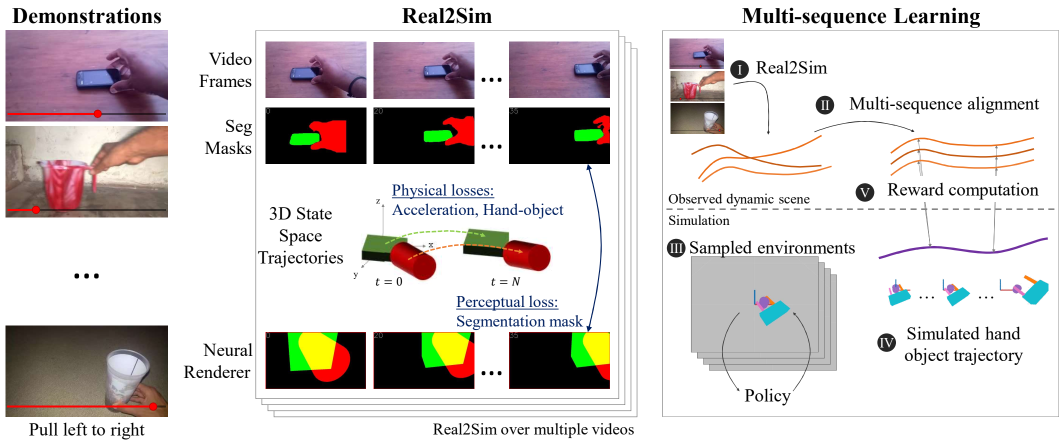

Figure 2: Overview of our method. **Demonstrations** (left): Our input is a few video demonstrations for each action, here depicting *pull left to right*. **Real2Sim** (center): Our optimization based approach obtains coarse 3D state representations for the hand (cylinder) and object (cuboid) based on a combination of physical losses modeling hand-object interactions and perceptual losses measuring similarity between the rendered image and a segmentation mask. **Multi-sequence learning** (right): I. Our approach estimates several trajectories, one for each video. II. We align these 3D trajectories in space and time. III. Environments with random object size and starting hand position are sampled. IV. The learning policy generates a simulated hand-object trajectory. V. We compute a dense reward that trains the agent to mimic the estimated trajectories.

our work, Shao et al. [37] present Concept2Robot that leverages an action classifier as a reward signal. They learn a single multi-task policy with a natural language instruction encoder that generalizes to small variations in actions. Nevertheless, the limited performance of the classifier, especially on simulated videos leads to noisy reward signals.

Mimicking actions. 3D human reconstruction [38] has seen use in teaching robots to mimic human actions [39, 40, 41]. While human pose can be extracted from a video with surprising accuracy, this is currently not the case for estimating the pose of 3D objects manipulated by hands. Our approximate scene representation addresses these challenges as it consists of coarse 3D blocks representing the main objects (and the hand) in the input video. In addition, our videos/demonstrations are aligned automatically by leveraging components of the state representation.

3 *Real2Sim*: Approximate State Estimation from Video

We model the structure and motion of the hand and object(s) in the video with coarse 3D models: a hand (wrist onwards) is represented as a cylinder with a fixed height and radius approximating the dimensions of a real hand, while objects are approximated as cuboids. In this section, we present our approach for 2D spatio-temporal video parsing, define the state space, and design perceptual and physics-based losses that optimize and estimate the states for a video.

Video pre-processing. We parse the video demonstrations using 2D visual reasoning tools: Mask-RCNN [6] for segmentation masks, and Hand-Object detector [5] that also provides a binary output to indicate touching or not. To ablate detection errors, we use ground-truth boxes by [25]. The final outputs of this module, obtained through a Kalman filter, include frame-level bounding boxes and segmentation masks for the hand and object(s). Please refer to the Appendix A.1 for details.

3.1 Approximate 3D state space

Consider an input video $\boldsymbol{v} = (v_1, \dots, v_T)$ with T frames, demonstrating how to perform some action, *e.g. pull left to right* (see Fig. 2). We define a state space to model the hand-object interaction, that mimics the visual content. (i) We assume the camera is fixed for each video and localize it in 3D using distance to origin, azimuth, and elevation. (ii) The hand is modeled as a cylinder ($40\,\mathrm{mm}$ radius and $150\,\mathrm{mm}$ height), with one of the edges symbolizing the manipulator/fingers. We use a 5D representation: $\boldsymbol{h} = (\boldsymbol{h}_p, \boldsymbol{h}_\theta)$, where $\boldsymbol{h}_p$ encodes the 3D position in Cartesian space and $\boldsymbol{h}_\theta$ captures two rotations: azimuth and elevation. (iii) Each object, represented as cuboid, is encoded by a 9D vector $\boldsymbol{o} = (\boldsymbol{o}_p, \boldsymbol{o}_s, \boldsymbol{o}_\theta)$ corresponding to 3D position in Cartesian coordinates $\boldsymbol{o}_p$, 3D object size $\boldsymbol{o}_s$, and the angle-axis formulation for the object rotation $\boldsymbol{o}_\theta$. (iv) Finally, we also encode whether the hand is touching the object as a binary label $\boldsymbol{\tau}$ for each frame. This leads to a $D = 18$ dimensional state representation (camera: 3, hand: 5, object: 9, touch: 1) for videos/actions with *one* object.

3.2 Losses and optimization

Our goal is to estimate a 3D state trajectory in $\mathbb{R}^{T\times D}$ over the T video frames that mimics the visual content of the demonstration. We employ two categories of loss functions, *perceptual* and *physical*. While the general problem of 2D to 3D has multiple solutions, two factors work in our favor: a fixed hand size helps estimate the distance between the camera and the hand; and a non-zero camera elevation that mimics ego-views helps estimates the depth of the objects placed on the surface.

Perceptual loss aims to estimate a 3D state trajectory that closely resembles, when projected, the video. We achieve this by representing the hand (cylinder) and objects (cuboids) as triangular meshes [42] that are rendered through a differentiable neural renderer [15], projecting the 3D state space of the hand and object to an image. We render binary object silhouettes without textures as they can be easily compared against the binary segmentation masks extracted from the video. As the optimization uses predicted segmentation masks, it can be considered as a self-supervised approach. The perceptual loss measures the reprojection error summed over all frames of the video for the hand and object(s) of interest in the scene:

$$L_{\text{perceptual}} = \sum\nolimits_{t=1}^{T} \left[\mathbb{1}(m_t^h) \cdot \|r_t^h - m_t^h\|^2 + \sum\nolimits_i \mathbb{1}(m_t^{o_i}) \cdot \|r_t^{o_i} - m_t^{o_i}\|^2 \right] , \tag{1}$$

where $m_t^h, m_t^{o_i}$ correspond to the predicted hand and ith object segmentation masks and $r_t^h, r_t^{o_i}$ correspond to rendered outputs. As frame segmentation or tracking may be unreliable, we make the estimation robust by applying the loss only when the mask is non-zero, indicated as $\mathbb{1}(\cdot)$. Additionally, a few missed detections are also interpolated using physics-based losses described next.

Physics based losses provide regularization and model the hand-object interaction. As part of the regularization, we minimize the acceleration (double-derivative in time) of all the object positions $\ddot{\boldsymbol{o}}_p$ and rotations $\ddot{\boldsymbol{o}}_\theta$, and the hand position $\ddot{\boldsymbol{h}}_p$ and rotation $\ddot{\boldsymbol{h}}_\theta$:

$$L_{\text{acc}} = \|\ddot{\boldsymbol{h}}_p\| + \|\ddot{\boldsymbol{h}}_\theta\| + \sum\nolimits_i \|\ddot{\boldsymbol{o}}_p^i\| + \|\ddot{\boldsymbol{o}}_\theta^i\| \, . \tag{2}$$

Acceleration loss smooths the trajectory and approximates the law of conservation of momentum. Additionally, as the object is assumed to be non-deformable, we also minimize the distance between the object size $\boldsymbol{o}_s$ and its mean over time $\bar{\boldsymbol{o}}_s$, $L_{\text{size}} = \sum_i \sum_t \|\boldsymbol{o}_s^i(t) - \bar{\boldsymbol{o}}_s^i\|$.

Finally, we regularize the hand-object interaction by imitating the law of inertia with infinite friction. We assume that only one object is manipulated at any time, and encourage the model to position the hand to be closer to the object if it is moving, and to minimize the object velocity otherwise:

$$L_{\text{interact}} = \boldsymbol{p}_{ho} \cdot \|\dot{\boldsymbol{o}}_p - \dot{\boldsymbol{h}}_p\| + (1 - \boldsymbol{p}_{ho}) \cdot \|\dot{\boldsymbol{o}}_p\| \, , \tag{3}$$

where $\boldsymbol{p}_{ho} = 1 - \sigma(\|\boldsymbol{h}_p - \boldsymbol{o}_p\|)$ is the probability that the hand is touching an object based on their positions, and $\sigma()$ is a parameterized sigmoid function that takes in a positive distance value and produces outputs in the range $[0, 1]$. The first term ensures that the hand and object move together when the hand touches the object and the second term penalizes object motion when the hand is not touching the object. Hand-object interactions are also encoded by the touch indicator $\boldsymbol{\tau}$ predicted by [5]. For the frames where the hand is said to touch the object, we set $\boldsymbol{h}_p = \boldsymbol{o}_p$.

Optimization. We use gradient descent to estimate the 3D state space trajectory that minimizes the total loss, a weighted combination of all terms: $\mathcal{L} = w_p L_{\text{perceptual}} + w_a L_{\text{acc}} + w_s L_{\text{size}} + w_i L_{\text{interact}}$. Owing to the differential neural renderer, we are able to backpropagate through the perceptual loss, while all other terms use standard differentiable components. See Appendix A.2 for details.

4 Learning Object Manipulation Policy from Multiple Videos

The objective of RL is to learn to manipulate object(s) so as to imitate the trajectories extracted from the videos. We use these trajectories to construct a reward function that trains the policy in a physics-based simulator mimicking the real world. To learn from multiple videos, trajectories are first spatio-temporally aligned. Note that, *across all actions, we use the same reward function* that encourages the policy to learn to mimic hand/object trajectories. This is a key aspect of our work as it does not require handcrafting a new reward for each action. Fig. 2 (right) presents an overview.

4.1 Spatio-temporal alignment of multiple extracted trajectories

Prior to using the trajectories from multiple demonstrations as part of the reward, we spatially align all trajectories of the same action by compensating for the camera orientation and the initial object position. After the alignment, all trajectories are expressed in the same reference frame given by the starting object position. In the case of multi-object actions we used the non-manipulated object for the alignment (see Fig. 6 in Appendix A.3).

The trajectories are also re-scaled in time such that all of them have the same duration and the action starts at a fixed timestamp. We estimate the *action phase*, the temporal part of the video where the actual action is performed by analyzing if the object is in motion (velocity $\dot{\boldsymbol{o}}_p$ above threshold) or if the hand is touching the object (predicted from [5]). An *approach phase* inserted at the beginning gives the gripper time to move to its initial position to execute the action. A *leave phase* at the end requires the gripper to be opened if the object should not remain grasped and the final position is set to be above the object position from the trajectory.

4.2 Simulator and learning setup

The simulated environment models the target robot environment that consists of a parallel jaw robotic gripper and cylindrical objects. The gripper pose is controlled by linear and angular velocity and the gripper closing is specified by the gap between gripper fingers. The simulation is performed for a fixed maximum horizon that corresponds to $10\,\mathrm{s}$ of simulated time and is prematurely reset if the gripper hits the ground. When the environment resets, the poses of the hand and object(s) could be set according to the first state of the trajectory. However, a policy learned with this constant starting state generalizes poorly to other starting conditions. Therefore, we randomize the gripper starting poses and the object sizes at the beginning of each episode. In particular, we employ automatic domain randomization, explained later in Sec. 5.2.

The goal of RL is to find a feed-forward policy that maps the observation to the distribution of actions that control the gripper. We represent the policy π as a Gaussian distribution: $\pi(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{s}), \Sigma)$, where $\boldsymbol{a} \in \mathbb{R}^7$ is the gripper command, $\boldsymbol{s} \in \mathbb{R}^8$ is the state containing time (1D) and gripper information (6D pose, 1D gripper opening), $\boldsymbol{\mu}(\boldsymbol{s}) \in \mathbb{R}^7$ is a mapping represented by a neural network (three hidden linear layers of size 128), and the diagonal covariance matrix $\Sigma \in \mathbb{R}^{7\times 7}$ is independent of the states. The Σ parameters and the neural network are trained by Proximal Policy Optimization [43] to maximize the expected discounted return defined as

$$R = \mathbb{E}\left[\sum\nolimits_{i=0}^{H} \gamma^i r(t_i)\right] , \tag{4}$$

where t_i is timestamp at step i, horizon H specifies the maximum number of time steps, r is the immediate reward that encodes the distance to trajectories extracted from the videos (presented next), and γ is the discount factor, set to 1 for our task to assign equal importance to future rewards.

4.3 Learning a policy from single or multiple videos

We start by defining the immediate reward for a single video as it forms the basis of multi-video learning. Note that such policies are often brittle as small errors in state estimation can lead to irrecoverable errors in the policy. For example, for the *push* action, we expect the hand to be behind the object. However, if this is not respected in the estimated trajectory, RL cannot recover from this error. For learning from a single video, the immediate reward r_v at time t is defined as:

$$r_v(t) = \sum_{\boldsymbol{q} \in \{\boldsymbol{h}_p, \boldsymbol{h}_\theta, \boldsymbol{o}_p^i, \boldsymbol{o}_\theta^i, \tau\}} w_q \exp\left(-\frac{1}{2} l_q d_q(\boldsymbol{q}(t), \tilde{\boldsymbol{q}}(t))\right) , \tag{5}$$

where $\boldsymbol{q}$ is one of the quantities (hand position, object orientation, etc.) extracted from the video, $\tilde{\boldsymbol{q}}$ is the corresponding quantity computed from the simulated environment, function $d(\cdot, \cdot)$ measures distance between those quantities, l_q are constant length-scales used to compensate for differences in units, and w_q are constant weights used to adjust the importance of different quantities. We assign the highest weight to object position as it is a crucial factor in judging the success of an action, and is difficult to learn as the policy may first need to learn other skills such as gripping the object.

We use different distance functions d_q to compare simulated and estimated trajectories: (i) squared euclidean norm for hand and object positions; (ii) squared angular distance for hand azimuth and elevation; (iii) squared angular distance between quaternions for object orientations; and (iv) the following weak signal to learn gripper closing:

$$d(\tau, \tilde{\tau}) = \begin{cases} \|\tau - \tilde{\tau}\|^2 & \text{if reference touch signal detected in the video, } \textit{i.e.}\ \tau = 1, \\ \infty & \text{otherwise}\,. \end{cases} \tag{6}$$

This ensures that a positive reward is provided for closing the gripper only when the hand is touching the object, and the robot may arbitrarily choose to open/close the gripper at other timesteps.

Reward for multiple videos. To learn a robust policy unaffected by errors in trajectory estimation, we propose to leverage multiple videos of the same action. We train on $\mathcal{S}_a$, the subset of demonstrations corresponding to an action a, by averaging aforementioned immediate reward for a single video (5):

$$r_a(t) = \frac{1}{|\mathcal{S}_a|} \sum\nolimits_{v \in \mathcal{S}_a} r_v(t)\,. \tag{7}$$

Note, that the average is computed over exponential rewards making it robust to outliers. Even when there are multiple ways to achieve the same task, our policy learns to mimic the most common trajectory and not the averaged trajectory. Rewards r_v (Eq. 5) and r_a (Eq. 7) are used as the immediate reward in Eq. (4) that is maximized by RL.

5 Evaluation

We first present an overview on the set of actions, a simulator-based benchmark, and metrics to evaluate the performance of learned policies. We then present an ablation study highlighting the impact of visual recognition tools and a curriculum learning approach. Finally, we compare policies learned on single or multiple videos and conclude with a brief summary of the real world robot setup.

5.1 Evaluation setup

Actions. We evaluate our approach on nine actions from the Something-Something action recognition dataset [7]: five single object manipulation tasks (*push/pull/pick*) and four two-object manipulation tasks (*put*). Fig. 3 illustrates one video per action category. For each action, six diverse videos in terms of backgrounds and manipulated object categories and shapes are selected.

Benchmark and metrics. We evaluate the robustness of trained policies by creating a benchmark of 1,000 samples that includes a variety of starting poses for the hand/gripper and sizes for the object(s). This includes challenging setups that are not similar to video demonstrations, *e.g.* the hand starts behind the object. We also create an easier benchmark with 1,000 samples drawn from a limited set of starting poses. All results are presented on the *hard* benchmark unless stated otherwise.

An estimated reward function is a proxy for the metric and is not always correlated with the successful execution of the action. Therefore, we use hand designed action-specific metrics (only during evaluation) that analyze the state of the hand and object while executing the action. Our stringent metric indicates a binary success/failure for each execution, *e.g. pull/push* actions check that the hand is in the correct position during object motion, that the object is standing, and it has moved at least $50\,\mathrm{mm}$ in the correct direction. Additional details are in Appendix B.

5.2 Experiments

Real2Sim setups. We analyze the impact of failures in visual parsing and obtain 3D state trajectories in three ways: A. ground-truth (GT) object/hand boxes [25] and GT action phase localization; B. GT boxes with predicted action phase; and C. predicted boxes with predicted action phase. Note that all setups still use predicted segmentation masks and automatic tracking. As the videos are crowd-sourced for action recognition, there is no GT for the 3D states. Thus, we use the action-specific metrics defined above to also compare the different Real2Sim setups. The average success rate over

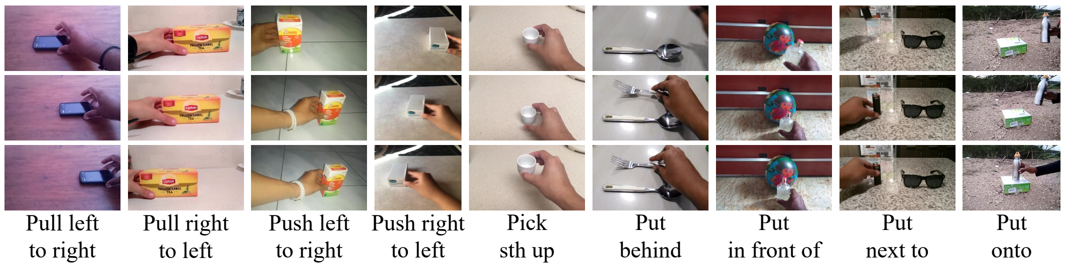

Figure 3: Example video demonstrations for the 9 actions used to evaluate our approach. Three frames for each video depict the beginning, middle, and the end of the annotated *action phase*.

all 54 videos for the three setups above is: A. 67%; B. 72%; and C. 61%. Surprisingly, method B works best, possibly due to the trajectory being estimated for the complete video followed by action phase localization. Nevertheless, as we will see later, policies learn actions best with setup A. Additional qualitative and quantitative analysis can be found in Appendix C.

Curriculum learning. We wish to learn policies that are robust to the initial gripper pose. This is normally achieved by randomizing the starting gripper position and orientation during training, *i.e.*, $\boldsymbol{h}_p \sim \mathcal{N}(\mathbf{0}, \sigma^2)$, $\boldsymbol{h}_\theta \sim \mathcal{U}(-180°, 180°)$, where $\mathcal{N}$ and $\mathcal{U}$ are Normal and Uniform distributions, and σ is set to $250\,\mathrm{mm}$. However, the blue bars in Fig. 4 show that the performance drops as σ increases indicating that learning with a randomized gripper pose is challenging. Inspired by Automatic Domain Randomization (ADR) [8], we design a curriculum learning strategy that linearly increases the randomness of the gripper pose during training. Policies that use ADR for both the 3D position and orientation (Fig. 4 orange) achieve the best results and are used in all other experiments.

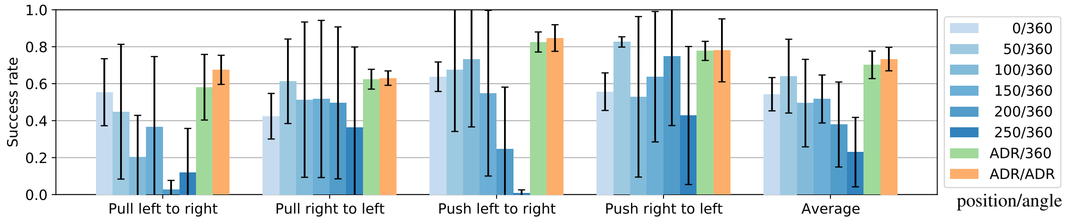

Figure 4: Success rate for multi-video policies trained with different randomization strategies using state trajectories estimated from GT boxes/GT action phase. **Legend**: the first number represents variability in the 3D position of the gripper, σ in mm; the second corresponds to the range of hand orientation in degrees. Five policies are trained for each parameter and we report mean and standard deviation (error bar in the plot).

Evaluating policies trained on different Real2Sim setups. Fig. 5 shows the success rate for each action for policies trained on multiple videos evaluated on the easy (top) and hard (bottom) benchmarks. While states predicted using method A (GT boxes and action phase, blue bars) result in higher performance, states that use method C (predicted boxes and action phase, green bars) degrade in performance by 10-15% on both benchmarks. Although method C results in higher performance for some actions (*e.g. put in front of*), on average, the estimated vision works slightly worse than ground-truth as expected. However, note that the action *put onto* is found to be particularly challenging as it requires precise modeling of objects, and about 5-7% of the gap between methods A and C is explained by this action alone. Overall, the trained policies are successful at executing the actions for a majority of initial states on both benchmarks. The gap between easy and hard benchmarks is also about 10-15%, indicating the challenges in starting position and strictness of our metric. A common failure case for our policy is when the gripper and object start close to each other and the gripper collides with the object while re-positioning in the approach phase.

Learning from single vs. multiple videos. We demonstrate the benefits of learning a policy from multiple videos by comparing them against policies trained on individual videos for all actions. In addition, we compare the proposed multi-video reward function to a baseline [39] that computes the reward by maximizing over the trajectories (instead of sum, see Eq. (7)). Table 1 shows that learning from single videos leads to a high variance in success rate due to the quality of the demonstration and state estimation process. In contrast, our proposed approach learns the action even if only a few trajectories are correct (see *put behind/in front of*). Finally, the multi-video policy used in the baseline (DeepMimic [39]) results in lower performance than our approach on most actions. The baseline computes maximum across trajectories, essentially selecting the easiest trajectory that would

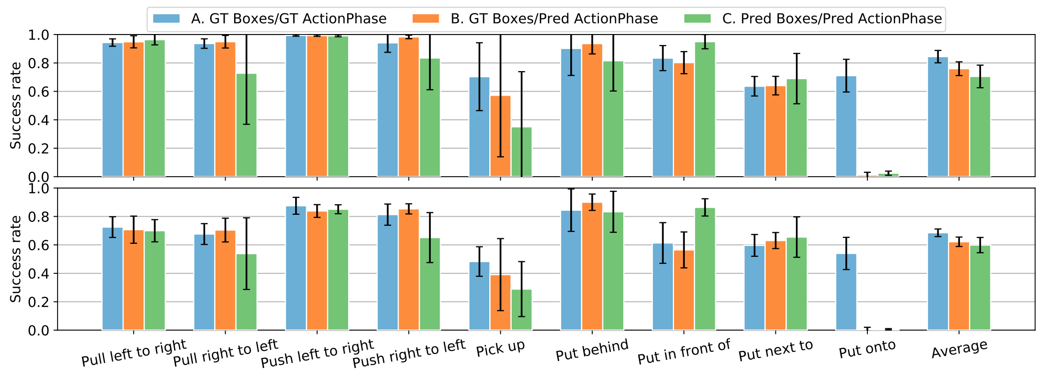

Figure 5: Success rate for policies trained on multiple videos with trajectories estimated from different Real2Sim setups. Top: easy benchmark; Bottom: hard benchmark. Each bar represents mean and std-dev over ten policies.

maximize reward. However, if this chosen trajectory does not successfully execute the action, it leads to negligible success rates. as is seen for the relatively easier actions of *pull left to right/right to left*.

Action	vid 1	vid 2	vid 3	vid 4	vid 5	vid 6	average	proposed	baseline [39]
Pull left to right	83	68	67	46	45	2	52	**66**	1
Pull right to left	62	58	52	13	0	0	31	**39**	0
Push left to right	88	83	67	57	41	0	56	**85**	0
Push right to left	85	85	73	71	60	58	72	**73**	71
Pick up	50	32	29	14	6	5	22	**38**	3
Put behind	86	85	57	45	35	29	56	**88**	82
Put in front of	94	80	54	46	35	18	54	**83**	65
Put next to	71	68	58	56	53	50	59	55	**78**
Put onto	19	0	0	0	0	0	**3**	0	0

Table 1: The success rate (in %) for single and multi-video policies and for the baseline [39]. Single video performances are sorted in descending order, and their average score is presented in the "average" column. Results are on the hard benchmark with states from method C, predicted boxes and action phase. Please refer to Appendix D.3 for similar results on the *easy* benchmark, and with states estimated from method A.

Transferring learned skills to the real robot. As the trained policy predicts the linear and angular velocity and the amount of gripper opening based on the current state of the gripper, we can compute the whole execution trajectory offline for an object with known initial pose (required for the alignment, and assumed known for our setup). We use numerical inverse kinematics tracking to compute the robot joint trajectory from the gripper Cartesian trajectory and quantize the gripper opening/closing due to lack of real-time gripper control capability for our Franka Emika Panda robot, shown in Fig. 1. Please refer to the supplementary video for a demo.

6 Conclusion

We presented an approach to teach robotic agents simple object manipulation skills by watching a few videos. We proposed a method that estimates a coarse 3D state representation for the hand and object(s) through a combination of 2D visual recognition, differentiable rendering, and an optimization method that learns from perceptual and physics based losses. These approximate state trajectories are used in an RL setup to successfully learn object manipulations for 9 single- and multi-object actions. We performed a thorough evaluation in a simulated environment, highlighting the benefits of adopting a curriculum learning strategy and learning from multiple videos, and also showed that the learned policies can be transferred to a real robot. Interesting future directions include incorporating physics into the Real2Sim estimation and scaling up to more actions.

Acknowledgment. This work was funded in part by the Louis Vuitton ENS Chair on Artificial Intelligence, 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), and the European Regional Development Fund under the project IMPACT (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000468).

References

[1] A. Miech, D. Zhukov, J.-B. Alayrac, M. Tapaswi, I. Laptev, and J. Sivic. HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In *ICCV*, 2019.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

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

[4] Y. Hasson, G. Varol, D. Tzionas, I. Kalevatykh, M. J. Black, I. Laptev, and C. Schmid. Learning joint reconstruction of hands and manipulated objects. In *CVPR*, 2019.

[5] D. Shan, J. Geng, M. Shu, and D. Fouhey. Understanding Human Hands in Contact at Internet Scale. In *CVPR*, 2020.

[6] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

[7] R. Goyal, S. E. Kahou, V. Michalski, J. Materzyńska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax, and R. Memisevic. The "something something" video database for learning and evaluating visual common sense. In *ICCV*, 2017.

[8] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[9] L. G. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[10] A. Gupta, A. A. Efros, and M. Hebert. Blocks World Revisited: Image Understanding Using Qualitative Geometry and Mechanics. In *ECCV*, 2010.

[11] G. Gkioxari, J. Malik, and J. Johnson. Mesh R-CNN. In *ICCV*, 2019.

[12] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. In *CVPR*, 2017.

[13] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. In *NeurIPS*, 2017.

[14] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV*, 2018.

[15] H. Kato, Y. Ushiku, and T. Harada. Neural 3D Mesh Renderer. In *CVPR*, 2018.

[16] S. Liu, T. Li, W. Chen, and H. Li. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. In *ICCV*, 2019.

[17] W. Chen, J. Gao, H. Ling, E. J. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *NeurIPS*, 2019.

[18] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. The EPIC-KITCHENS Dataset: Collection, Challenges and Baselines. *PAMI*, 2020.

[19] B. Tekin, F. Bogo, and M. Pollefeys. H+O: Unified Egocentric Recognition of 3D Hand-Object Poses and Interactions. In *CVPR*, 2019.

[20] M. Höll, M. Oberweger, C. Arth, and V. Lepetit. Efficient Physics-Based Implementation for Realistic Hand-Object Interaction in Virtual Reality. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2018.

[21] X. Tang, X. Hu, C.-W. Fu, and D. Cohen-Or. GrabAR: Occlusion-aware Grabbing Virtual Objects in AR. In *ACM User Interface Software and Technology Symposium*, 2020.

[22] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black. Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image. In *ECCV*, 2016.

[23] J. Romero, D. Tzionas, and M. J. Black. Embodied Hands: Modeling and Capturing Hands and Bodies Together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6):245:1–245:17, 2017.

[24] Y. Zhou, M. Habermann, W. Xu, I. Habibie, C. Theobalt, and F. Xu. Monocular Real-time Hand Shape and Motion Capture using Multi-modal Data. In *CVPR*, 2020.

[25] J. Materzynska, T. Xiao, R. Herzig, and H. Xu. Something-Else: Compositional Action Recognition with Spatial-Temporal Interaction Networks. In *CVPR*, 2020.

[26] S. Hampali, M. Rad, M. Oberweger, and V. Lepetit. HOnnotate: A method for 3D Annotation of Hand and Object Poses. In *CVPR*, 2020.

[27] Y. Hasson, B. Tekin, F. Bogo, I. Laptev, M. Pollefeys, and C. Schmid. Leveraging Photometric Consistency over Time for Sparsely Supervised Hand-Object Reconstruction. In *CVPR*, 2020.

[28] M. Hazara and V. Kyrki. Reinforcement learning for improving imitated in-contact skills. In *International Conference on Humanoid Robots*, 2016.

[29] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv: 1707.08817*, 2017.

[30] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *CoRL*, 2019.

[31] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee. Kinesthetic teaching and attentional supervision of structured tasks in human-robot interaction. *Autonomous Robots*, 43:1291–1307, 2019.

[32] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid. Learning to combine primitive skills: A step towards versatile robotic manipulation. In *ICRA*, 2020.

[33] P. Sermanet, K. Xu, and S. Levine. Unsupervised Perceptual Rewards for Imitation Learning. In *RSS*, 2017.

[34] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, and S. Levine. Time-Contrastive Networks: Self-Supervised Learning from Video. In *ICRA*, 2017.

[35] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from Observation: Learning to imitate behaviors from raw video via context translation. In *ICRA*, 2018.

[36] L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine. AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos. In *RSS*, 2020.

[37] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg. Concept2Robot: Learning Manipulation Concepts from Instructions and Human Demonstrations. In *RSS*, 2020.

[38] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik. End-to-end Recovery of Human Shape and Pose. In *CVPR*, 2018.

[39] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, 2018.

[40] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine. SFV: Reinforcement Learning of Physical Skills from Videos. *ACM Trans. Graph.*, 37(6), 2018.

[41] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. In *RSS*, 2020.

[42] Dawson-Haggerty et al. trimesh, 2019. URL `https://trimsh.org/`.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, 2017.

[44] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, 2014.

[45] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple Online and Realtime Tracking. In *ICIP*, 2016.

Appendix

We present additional details for our method and a variety of qualitative and quantitative results. This appendix is complemented by additional material including a video describing the contributions in brief and demonstrating results on a real robot, and two HTML files: `real2sim.html` and `policies.html` showcasing qualitative real2sim and policy results respectively.

A Implementation Details

We present some additional implementation details for (i) *Video pre-processing* that estimates hand and object detections and segmentation masks; (ii) *Real2Sim*, that estimates 3D state trajectories from videos; and (iii) *RL* that learns a policy to mimic object manipulation based on extracted trajectories.

A.1 Video pre-processing

We parse each video with a frame-level hand-object detector and semantic segmentation method.

(i) Mask-RCNN [6] predicts segmentation masks for multiple COCO categories [44], including the *person* label that segments the hand, and various objects that may appear in the video.

(ii) Hand-Object detector [5] is a recent work that detects hands, objects, and their interactions (touching or not). As an alternative to automatic detection, we also analyze the impact of using ground-truth boxes (Something-Else [25]). However, note that this still uses automatic segmentation to convert boxes to pixel-level masks.

Tracking. We track the hand and object(s) of interest by adopting a Kalman-filter [45] that links frame-level detections using a simple intersection-over-union overlap based metric. Missed detections are ignored, and the segmentation mask is set to all zero. The physics-based losses, especially the acceleration loss, interpolates across missed detections by preventing rapid changes in position and orientation.

A.2 Real2Sim

3D state estimation parameters. States are represented as learnable parameters (real numbers) that are normalized through sigmoid or tanh functions.

We initialize 3D state parameters of the hand (3D position, elevation) and the object (3D size and 3D position) by sampling from a normal distribution: $0.1 \cdot \mathcal{N}(0, 1)$. Hand azimuth is initialized from $0.1 \cdot \mathcal{N}(\pi/2, 1)$ to model the hand orientation in egocentric videos (pointing away from the camera at 90°), and finally object rotation is initialized as $0.01 \cdot \mathcal{N}(0, 1)$ as we expect objects to be upright.

We normalize these parameters as follows: (i) object sizes are normalized through a sigmoid function in range $0\,\mathrm{mm}$ to $300\,\mathrm{mm}$; (ii) object position is normalized through a tanh function in range $-1200\,\mathrm{mm}$ to $1200\,\mathrm{mm}$; (iii) hand position is normalized through a tanh function in range $-1500\,\mathrm{mm}$ to $1500\,\mathrm{mm}$; and (iv) hand elevation is normalized using a tanh function in range $-90°$ to $90°$. We found that normalizing hand azimuth makes state estimation challenging, possibly due to $0°$ and $360°$ corresponding to the same angle. The object position in z direction is set to half the object size in z direction as objects may lie on the table.

Weights for the loss terms are as follows. For the perceptual loss $w_p = 0.3$, acceleration term $w_a = 5$, interaction term $w_i = 1$, and object size term $w_s = 1000$. Note that the high weight w_s ensures that the object size does not change over the estimated trajectory.

Learning details. We use the Adam optimizer with a (considerably high) learning rate 10^{-2} to update the state parameters. All states for the entire video are updated at each iteration as this allows imposing acceleration based regularization at every frame. The parameters are updated for 400 iterations (most parameters converge typically around 200-250 iterations). The models are implemented using PyTorch.

Action phase estimation. Given the entire 3D state trajectory, we use two cues to predict when the action is taking place. First, we consider whether the hand is touching the object [5]. Second, we consider whether the object is in motion, by checking if the velocity is greater than $70\,\mathrm{mm\,s^{-1}}$. These predictions are quite reliable with an average intersection-over-union of 0.81.

A.3 Learning object manipulation policy from multiple videos

Spatial alignment. We spatially align all trajectories of the same action by compensating for the camera orientation and the initial object position as shown in Fig. 6. The spatial alignment merely resets the frame-of-reference (camera and initial object position) to an object in the scene, something that can be achieved for most actions.

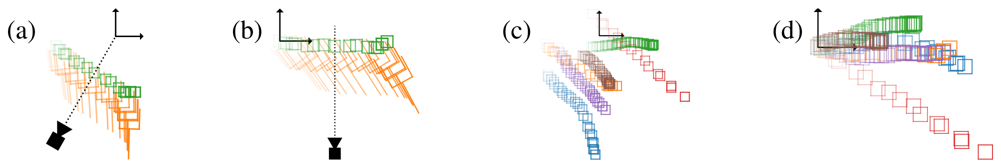

Figure 6: Top-view explaining the spatial alignment of extracted trajectories for the action *pull left to right*. Time is indicated by transparency, the most transparent color represents the start of the trajectory. For one video, **(a)** shows the estimated camera pose, hand (orange), and object (green) trajectories, while **(b)** shows the trajectories after alignment, compensating for camera pose and initial object position. **(c)** depicts object trajectories extracted from multiple videos of the same action, and **(d)** presents their aligned version.

Reinforcement learning parameters. Reward weights required for computation of Eq. (5) are shown in Table 2. We use PPO [43] algorithm that collects 70 episodes per policy update. Policy update uses learning rate $3 \cdot 10^{-4}$ and is performed for 25 epochs, value 0.2 for likelihood ratio clipping threshold, value 100 for gradient clipping, weight $1 \cdot 10^{-8}$ for entropy loss and generalized advantage estimation with lambda set to 0.95. Discount factor is set to 1. Total number of policy updates is set to 1,000.

For Automatic Domain Randomization (ADR), we linearly increases randomness for gripper orientation from the 1st until the 400th policy update and for gripper position from 500th until the 900th policy update. The policy is represented by a fully connected neural network with three hidden layers each containing 128 neurons. The models are implemented using PyTorch and *rlpyt*, a RL library for PyTorch.

Parameter	$\boldsymbol{h}_p$	$\boldsymbol{h}_\theta$	$\boldsymbol{o}_p^i$	$\boldsymbol{o}_\theta^i$	τ
w_q	0.2	0.2	0.5	0.05	0.05
l_q	100	10	100	10	10000

Table 2: Reward parameters for RL.

B Benchmark and Metrics

The *easy* and *hard* benchmarks are sampled randomly from distributions specified in Table 3. The random sample is discarded if a collision is detected among any two of hand, object(s), or ground plane. In total 1,000 collision free samples are obtained for each benchmark and used for the evaluation.

Metrics. We design an evaluation metric for each action separately by analyzing the positions and orientations of the hand and object trajectories. For *pull* and *push* actions, the metric requires that object is moved in the correct direction by at least $50\,\mathrm{mm}$. During the object motion, the hand has to be oriented in the direction of the motion for push and in a reversed direction for pull actions (see Fig. 7). Note that we also require the object stay upright (not fall) during the pull or push actions. For *pick something up* (not visualized), the metric checks that the object is held at least $10\,\mathrm{mm}$ above the ground by the gripper. Finally, for the two object actions (*put*), we consider the final state of the objects. The final position of the manipulated object should be within a $120°$ arc around the static object, and within $500\,\mathrm{mm}$ radius from the center of the static object. The proposed metric is binary, and evaluates whether the policy executes the action for a given starting hand position and object size.

Parameter	*easy* benchmark	*hard* benchmark
gripper position x	$\mathcal{U}(-100\,\text{mm}, 100\,\text{mm})$	$\mathcal{U}(-250\,\text{mm}, 250\,\text{mm})$
gripper position y	$\mathcal{U}(-100\,\text{mm}, 0\,\text{mm})$	$\mathcal{U}(-250\,\text{mm}, 250\,\text{mm})$
gripper position z	$\mathcal{U}(100\,\text{mm}, 200\,\text{mm})$	$\mathcal{U}(0\,\text{mm}, 250\,\text{mm})$
gripper azimuth	$\mathcal{U}(-180°, 180°)$	$\mathcal{U}(-180°, 180°)$
gripper elevation	$\mathcal{U}(0°, 80°)$	$\mathcal{U}(-80°, 80°)$
1st object position	$\mathbf{0}$	$\mathbf{0}$
1st object orientation	upright	upright
1st object radius	$\mathcal{U}(40\,\text{mm}, 50\,\text{mm})$	$\mathcal{U}(40\,\text{mm}, 50\,\text{mm})$
1st object height	$\mathcal{U}(60\,\text{mm}, 80\,\text{mm})$	$\mathcal{U}(40\,\text{mm}, 100\,\text{mm})$
2nd object pose	in hand	in hand
2nd object radius	$\mathcal{U}(40\,\text{mm}, 50\,\text{mm})$	$\mathcal{U}(40\,\text{mm}, 50\,\text{mm})$
2nd object height	$\mathcal{U}(60\,\text{mm}, 80\,\text{mm})$	$\mathcal{U}(40\,\text{mm}, 100\,\text{mm})$

Table 3: Ranges for benchmark generation. Symbol $\mathcal{U}$ represents uniform distribution.

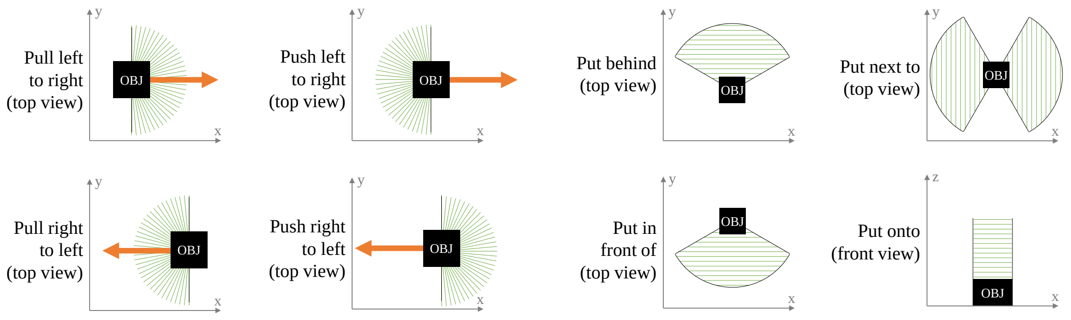
Pull left to right (top view)
Push left to right (top view)
Put behind (top view)
Put next to (top view)
Pull right to left (top view)
Push right to left (top view)
Put in front of (top view)
Put onto (front view)
OBJ
x
y
z

Figure 7: Visualization of the metrics used to evaluate whether the action is performed correctly. Note how the hand is in the direction of motion for pull actions, while in the opposite area for the push actions. Two object actions allow the object to be *put* in a specific cone of area *behind/in front of/next to* the object. For *put onto*, we show the front view, indicating that the second object should be placed above the static one. Please refer to the text for exact numerical details.

C Additional Results: Real2Sim

Quantitative results for different ablation methods for Real2Sim were presented as summaries in the main paper. We elaborate on those in Table 4, showing the fraction of correctly extracted trajectories for each action and method. Recall, we analyze performance based on three methods depending on whether automatic or ground-truth was used for performing spatio-temporal localization: A. ground-truth object/hand boxes and ground-truth action phase localization; B. ground-truth boxes with predicted action phase; and C. predicted boxes with predicted action phase. We observe that method B works best, possibly due to the trajectory being estimated for the complete video followed by truncation of states using predicted action phase localization. In contrast, for method A, we first truncate the video and then perform state estimation.

Method	Pull left to right	Pull right to left	Push left to right	Push right to left	Pick up	Put behind	Put in front of	Put next to	Put onto	**Total**
A	6/6	4/6	5/6	6/6	2/6	2/6	5/6	4/6	2/6	36/54
B	6/6	5/6	6/6	6/6	3/6	2/6	5/6	6/6	0/6	39/54
C	5/6	4/6	5/6	6/6	2/6	1/6	4/6	6/6	0/6	33/54

Table 4: Benchmarking states.

Visualization. The metrics provide a limited understanding of the state estimation process (see Sec. 5.2). We visualize the estimated hand and object trajectories from a top-view perspective in Fig. 8. To obtain better insights, we visualize additional results as GIFs in an HTML page `real2sim.html` showing all 6 videos for each of the 9 actions. In each row of the table, we show the full video,

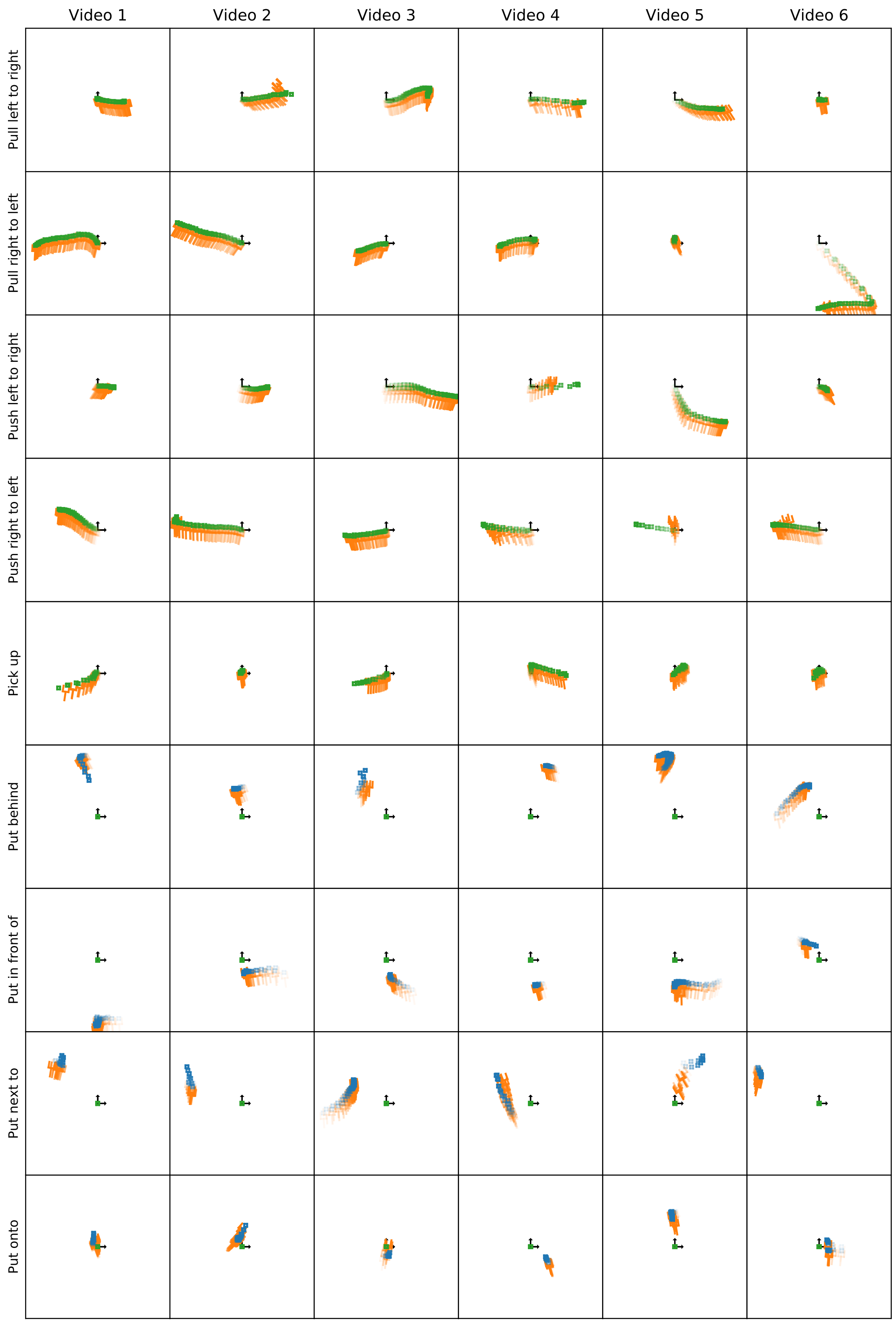

Figure 8: Top view for reconstructed states of hand (orange) and objects (green, blue) in time using method C. Transparency indicates temporal progression, more transparent means older in time.

followed by segmentation masks $m_t^h, m_t^{o_i}$ and corresponding 3D state renderings via the neural renderer $r_t^h, r_t^{o_i}$. All video ids correspond to the original ids in the Something-Something dataset. The estimations from all three methods are shown as three columns of the table.

D Policy Training Results

We present additional results and analysis for training a policy with different settings.

D.1 Curriculum learning

In the main paper, we looked at how curriculum learning with Automatic Domain Randomization (ADR) helped our agent to learn a better policy. While the main paper presented results on the *hard* benchmark (see Fig. 4), here, we present additional results on the *easy* benchmark in Fig. 9. In particular, we observe the same trends. The blue bars in Fig. 9 show that the performance drops as σ increases indicating that learning with a randomized gripper pose is challenging. Our strategy to use ADR for both the 3D position and orientation (Fig. 9 orange) achieves the best results and is used in all other experiments.

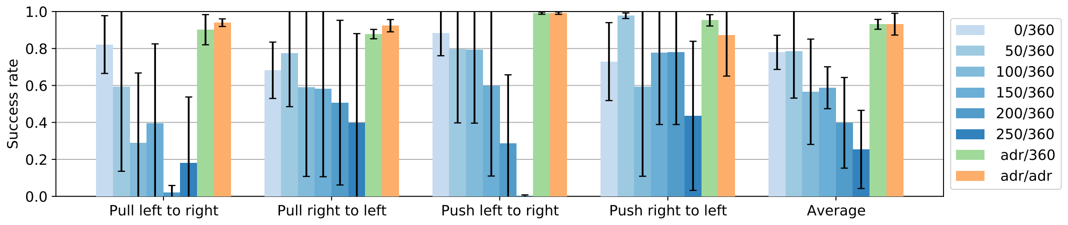

Figure 9: Success rate for multi-video policies trained with different randomization strategies using state trajectories from method A, and evaluated on the *easy* benchmark. **Legend**: the first number represents variability in the 3D position of the gripper, σ in mm; the second corresponds to the range of hand orientation in degrees. Five policies are trained for each parameter and we report mean and standard deviation (error bar in the plot).

D.2 Impact of GT in Real2Sim setups

We present additional analysis of the difference in performance due to the use of GT bounding boxes or GT action phase. Recall that these are referred to as method A, B, and C in the main paper. On average, Fig. 5 shows that the estimated vision works slightly worse than ground-truth as expected. Actions *put behind* and *put onto* follow this trend. For *put next to*, the performance is within error bars. By looking at `real2sim.html` (supplementary), we see that there are a few demonstrations where Real2Sim could be improved with better object segmentation and tracking. We think these individual instances are outliers and are not representative of estimated vision working better, even for *put in front of*. Referring to `real2sim.html` method A, we see that video 5 is challenging due to the elongated scissors, while video 6 fails as the object is placed too far for *put next to*. For *put in front of*, video 1 places the object too far, and video 2 fails probably due to the GT box extending beyond the object mask.

Big performance gap for *put onto*. We observe that method A achieves 60% success rate for *put onto*, while methods B and C are close to 0%. This can be explained by analyzing the GIFs for video 2 and video 4 in `real2sim.html`, where states estimated by method A are better than method C. Quantitatively, method A gets 2/6 correct, while method C gets none (Table 4). RL is able to use these 2 trajectories to learn the task. Having ground-truth 2D boxes alleviates significant confusion while tracking the two objects. Additionally, the big gap is also likely due to the binary nature of the metric that requires balancing objects: it either works, or the object falls, there is no half-way success.

D.3 Learning from single vs. multiple videos.

Similar to Table 1 of the main paper that shows the performance of learning from single vs. multiple videos on the *hard* benchmark with states obtained from method C, we present quantitative performance in a few more different settings.

In particular, Table 5 shows the performance on the *easy* benchmark when using states estimated from method A. Our agent is able to learn to perform most actions, often with success rates greater than 90% (5/9 actions). Table 6 presents the results on using the same states, but on the *hard* benchmark. We notice a drop in performance for all actions, particularly for starting positions of the hand being away from the camera. Finally, Table 7 showcases the performance when using states estimated from method C on the *easy* benchmark.

Overall, we observe that our proposed multi-video learning method outperforms learning from single videos, as some demonstrations may be noisy. In addition, using sum (see Eq. (7)) instead of max [39] shows consistent performance improvements across all experiments.

Action	vid 1	vid 2	vid 3	vid 4	vid 5	vid 6	average	proposed	baseline
Pull left to right	97	89	98	95	83	0	77	93	53
Pull right to left	57	64	94	0	94	0	51	91	95
Push left to right	99	99	62	98	98	0	76	99	33
Push right to left	98	84	99	0	99	99	80	92	93
Pick up	83	43	64	55	97	0	57	88	49
Put behind	99	99	94	5	89	0	64	95	95
Put in front of	75	20	60	78	79	74	64	79	76
Put next to	98	72	59	98	45	69	74	67	75
Put onto	0	89	0	78	0	1	28	73	0

Table 5: The success rate (in %) for single and multi-video policies and for the baseline [39]. Single video performances are averaged in the "average" column. Results are on the *easy* benchmark with states from method A.

Action	vid 1	vid 2	vid 3	vid 4	vid 5	vid 6	average	proposed	baseline
Pull left to right	71	61	76	65	62	1	56	62	35
Pull right to left	39	47	74	2	64	0	37	60	76
Push left to right	90	72	31	86	86	0	61	87	27
Push right to left	84	59	90	3	84	89	68	81	79
Pick up	37	36	45	45	57	7	38	57	36
Put behind	91	89	89	31	90	0	65	86	88
Put in front of	45	15	41	52	59	55	44	56	51
Put next to	88	76	65	94	45	55	70	65	72
Put onto	0	65	0	58	0	0	20	57	0

Table 6: The success rate (in %) for single and multi-video policies and for the baseline [39]. Single video performances are averaged in the "average" column. Results are on the *hard* benchmark with states from method A.

Action	vid 1	vid 2	vid 3	vid 4	vid 5	vid 6	average	proposed	baseline
Pull left to right	99	88	88	57	79	0	68	93	0
Pull right to left	90	81	83	19	0	0	45	62	0
Push left to right	98	99	93	74	55	0	70	99	0
Push right to left	99	89	95	89	82	86	90	91	93
Pick up	75	43	17	12	0	0	24	64	0
Put behind	87	82	33	21	5	4	38	87	79
Put in front of	99	88	56	76	61	31	68	92	84
Put next to	74	71	59	69	50	47	62	53	77
Put onto	25	3	3	1	0	0	5	1	1

Table 7: The success rate (in %) for single and multi-video policies and for the baseline [39]. Single video performances are averaged in the "average" column. Results are on the *easy* benchmark with states from method C.

Visualization. Similar to the results for Real2Sim, we create another HTML page, `policies.html` showing results of trained policies. For each action, we present one successful and one failure example in the simulator, and the result of transferring the policy to a real robot.