Asymmetric Actor Critic for Image-Based Robot Learning

Lerrel Pinto
Carnegie Mellon University
lerrelp@cs.cmu.edu

Marcin Andrychowicz
OpenAI
marcin@openai.com

Peter Welinder
OpenAI
pw@openai.com

Wojciech Zaremba
OpenAI
woj@openai.com

Pieter Abbeel
OpenAI
pieter@openai.com

***Abstract*—Deep reinforcement learning (RL) has proven a powerful technique in many sequential decision making domains. However, robotics poses many challenges for RL, most notably training on a physical system can be expensive and dangerous, which has sparked significant interest in learning control policies using a physics simulator. While several recent works have shown promising results in transferring policies trained in simulation to the real world, they often do not fully utilize the advantage of working with a simulator. In this work, we propose the Asymmetric Actor Critic, which learns a vision-based control policy while taking advantage of access to the underlying state to significantly speed up training. Concretely, our algorithm employs an *actor-critic* training algorithm in which the critic is trained on full states while the actor (or policy) is trained on images. We show that using these asymmetric inputs improves performance on a range of simulated tasks. Finally, we combine this method with domain randomization and show real robot experiments for several tasks like picking, pushing, and moving a block. We achieve this simulation to real-world transfer without training on any real-world data. Videos of these experiments can be found in www.goo.gl/b57WTs.**

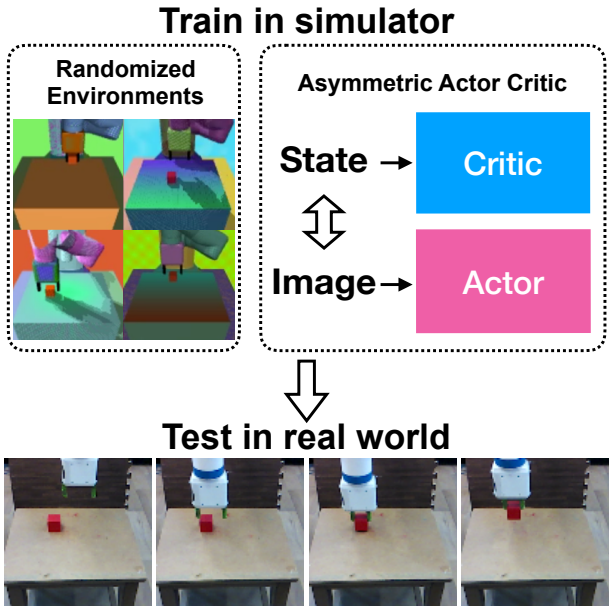

Fig. 1. By training policies with asymmetric inputs for actor-critic along with domain randomization, we learn complex visual policies that can operate in the real world without having seen any real world data in training.

I. INTRODUCTION

Reinforcement learning (RL) coupled with deep neural networks has recently led to successes on a wide range of control problems, including achieving superhuman performance on Atari games [26] and beating the world champion in the classic game of Go [45]. In physics simulators, complex behaviours like walking, running, hopping and jumping have also been shown to emerge [43, 23].

However, in the context of robotics, learning complex behaviours faces two unique challenges: scalability and safety. Robots are slow and expensive, which makes existing data intensive learning algorithms hard to scale. These physical robots could also damage themselves and their environment while exploring these behaviours. A recent approach to circumvent these challenges is to train on a simulated version of the robot and then transfer the learned policy to the real robot [8, 12, 18, 39, 7, 52, 40, 13].

One common method for robot learning using simulators is by training a visual policy on rendered images and then transferring it to a real robot [40, 39, 47]. These visual policies allow for easy instrumentation on the robot's end. However, this comes at the increased cost of learning from high dimensional visual observations. Other methods first learn state-based policies in the simulator. Then, these state-based policies can be behaviour cloned or imitation learned to a visual policy [15]. This allows for faster learning, but behaviour cloning does not work well when the observation domain is different between the expert and the cloned policy (see Section. V-D). Instead of behaviour cloning, one could also perform explicit state estimation from visual observations and employ a state based policy. However, as demonstrated in Andrychowicz et al. [3] the errors in estimation often leads to failure with complex policies. This leads us to a conundrum, i.e., training on images is hard because of their high dimensionality, on the other hand learned state-based policies are hard to use on a robot.

We solve this conundrum by learning a policy that operates only on visual observations (RGBD images), but accesses full state during training. Physics simulators give us access to both the full state of the system and the rendered images of the scene. But how can we combine these observations to train complex behaviours faster? In this work, we exploit the access to states and train an *actor-critic* algorithm [19, 23] that uses asymmetric inputs, i.e. the actor takes visual partial

observations as input while the critic takes the underlying full state as input. Since the critic works on the low dimensional full state, it learns the *state-action* value function much faster. This also allows for better updates for the actor. During testing, the actor is employed on the high dimensional visual observations and does not depend on the full state (the full state is only used during training). This novel technique allows us to train an actor/policy network on visual observations while exploiting the availability of full states to train the critic.

However, simulators are not perfect representations of the real world. The domain of observations in the real world (real camera images) significantly differs from rendered images from a simulator. This makes directly transferring policies from the simulator to the real world difficult. However, Sadeghi and Levine [40] and other researchers [47, 49] show how randomizing textures and lighting allows for effective transfer to the real world. We employ these domain randomization techniques to transfer to a real robot.

Our experimental evaluations on 2D environments such as *Particle* and *Reacher* and 3D robot environments such as *Fetch Pick* show that Asymmetric Actor Critic significantly improves over both learning with only visual observations [23] and cloning from a state based expert [15]. By combining our Asymmetric Actor Critic training with domain randomization [47], we show that these visual policies can *Pick*, *Push* and *Move* a block using a real robot without any training on the physical system (see Figure 1).

In summary, this paper presents three main contributions: (a) We propose a novel technique for learning visual policies by exploiting low dimensional full state observations in a simulator. (b) We show that this asymmetric training significantly speeds up training. (c) We can use these policies on real robots without any training on the physical system.

II. RELATED WORK

A. Reinforcement Learning

Recent work in deep reinforcement learning (deep RL) has shown impressive results in the domain of games [26, 45] and simulated control tasks [43, 23]. The class of RL algorithms our method employs are called *actor-critic* algorithms [19]. Deep Deterministic Policy Gradients (DDPG) [23] is a popular *actor-critic* algorithm that has shown impressive results in continuous control tasks. Although we use DDPG for our base optimizer, our method is applicable to arbitrary *actor-critic* algorithms.

Learning policies in an environment that provides only sparse rewards is a challenging problem due to a very limited feedback signal. However it has been shown that sparse rewards often allow for better policies when trained appropriately [3]. Moreover having sparse rewards allows us to circumvent manual shaping of the reward function.

Foerster et al. [10] and Lowe et al. [24] have trained *actor-critic* algorithms for multi-agent environments. Here a single critic is trained with information from all the agents while multiple actors are trained for each agent using each agent's observation. However we use asymmetric inputs in the context of speeding up learning of visual policies.

B. End-to-end robot learning

Learning policies directly from observations [21, 28, 2, 22] has received significant attention over the last few years. These works have shown that without explicit modeling and state estimation, complex manipulation behaviors can be learned. Reducing the error-prone aspects of modeling and estimation also allows for easier deployment of robots. However, these end-to-end learning methods often require significant amounts of real world robot data [28, 2, 22] which makes it difficult to scale and dangerous to the environment. Simulators hold promise in alleviating both these challenges since simulators are orders of magnitude faster than the real world and do not perform physical interactions. This paper looks to improve robot learning by training completely in a simulator without seeing any real world robot data.

C. Transfer from simulation to the real world

Bridging the reality gap in transferring policies trained in a simulator to the real world is an active area of research in the robot learning community. Apart from being faster, simulators are also safer than real robots [11, 13]. One approach to transfer simulator policies to the real world is to make the simulator as close to the real world as possible [14, 33, 37]. But these methods have had limited success due to hard system identification problems.

Another approach is domain adaptation from the simulator [8, 12, 18, 39, 7, 52], since it may be easier to fine tune from a simulator policy than training in the real world. However if the simulator differs from the real world, the policy trained in simulation can perform very poorly in the real world and fine tuning may not be any easier than training from scratch. This limits most of these works to learning simple behaviours. Making policies robust for physics adaptation [36, 32, 51] is also receiving interest, but these methods haven't been shown to be powerful enough to work on real robots. Using bottlenecks [52] has been shown to help domain adaptation for simple tasks like *reaching*. In this work, we show how bottlenecks can be exploited for more complex fine manipulation tasks. We also note that our method is complementary to *actor-critic* versions of several of these domain adaptation methods.

A promising approach is domain adaptation by domain randomization [47, 40]. Here the key idea is to train on randomized renderings of the scene, which supports learning robust policies for transfer. However these works do not show transfer to precise manipulation behaviours. We show that this idea of randomization can be extended to complex behaviours when coupled with our asymmetric actor critic.

D. Robotic tasks

We perform real robot experiments on tasks like picking, pushing, and moving a block. The *Picking* task is similar to grasping objects [6, 28, 31, 25], however in this work we learn

a visual end-to-end policy that moves to the object, grasps it and moves the grasped object to its desired position. The focus is on learning the fine manipulation behaviour directly from visual inputs. The tasks of *Forward Pushing* and *Block Move* are similar to pushing objects [5, 2, 29, 30], however as in the case of *Picking*, this paper focuses on the visual learning of the fine pushing behaviour.

III. BACKGROUND

Before we discuss our method, we briefly introduce some background and formalism for the RL algorithms used. A more comprehensive introduction can be found in Kaelbling et al. [16] and Sutton and Barto [46].

A. Reinforcement Learning

In this paper we deal with continuous space Markov Decision Processes that can be represented as the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, r, \gamma, \mathbb{S})$, where $\mathcal{S}$ is a set of continuous states and $\mathcal{A}$ is a set of continuous actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the transition probability function, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, γ is the discount factor, and $\mathbb{S}$ is the initial state distribution. $\mathcal{O}$ is a set of continuous partial observations corresponding to states in $\mathcal{S}$.

An episode for the agent begins with sampling s_0 from the initial state distribution $\mathbb{S}$. At every timestep t, the agent takes an action $a_t = \pi(s_t)$ according to a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. At every timestep t, the agent gets a reward $r_t = r(s_t, a_t)$, and the state transitions to s_{t+1}, which is sampled according to probabilities $\mathcal{P}(s_{t+1}|s_t, a_t)$. The goal of the agent is to maximize the expected return $E_{\mathbb{S}}[R_0|\mathbb{S}]$, where the return is the discounted sum of the future rewards $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$. The Q-function is defined as $Q^{\pi}(s_t, a_t) = E[R_t|s_t, a_t]$. In the partial observability case, the agent takes actions based on the partial observation, $a_t = \pi(o_t)$, where o_t is the observation corresponding to the full state s_t.

B. Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradients (DDPG) [23] is an *actor-critic* RL algorithm that learns a deterministic continuous action policy. The algorithm maintains two neural networks: the policy (also called the actor) $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ (with neural network parameters θ) and a Q-function approximator (also called the critic) $Q^{\pi}_{\phi} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (with neural network parameters ϕ).

During training, episodes are generated using a noisy version of the policy (called behaviour policy), e.g. $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$, where $\mathcal{N}$ is the Normal distribution noise. The transition tuples (s_t, a_t, r_t, s_{t+1}) encountered during training are stored in a replay buffer [26]. Training examples sampled from the replay buffer are used to optimize the critic. By minimizing the Bellman error loss $\mathcal{L}_c = (Q(s_t, a_t) - y_t)^2$, where $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$, the critic is optimized to approximate the Q-function. The actor is optimized by minimizing the loss $\mathcal{L}_a = -E_s[Q(s, \pi(s))]$. The gradient of $\mathcal{L}_a$ with respect to the actor parameters is called the deterministic policy gradient [44] and can be computed by backpropagating through the combined critic and actor networks. To stabilize the training, the targets for the actor and the critic y_t are computed on separate versions of the actor and critic networks, which change at a slower rate than the main networks. A common practice is to use a Polyak averaged [34] version of the main network.

In this *actor-critic* framework, the role of the critic is to improve the policy gradient estimates by providing a value function (Q) estimate. The only way the critic interacts with the learning is through the value estimate. The more accurate the value estimate, the better the policy gradient is estimated. In Asymmetric Actor Critic this critic is trained on state information, which allows the value estimates to be more accurate compared to a critic trained on images. Furthermore, since the policy only depends on value estimates, a visual policy can be learned.

C. Multigoal RL

We are interested in learning policies that can achieve multiple goals (a universal policy). One way of doing this is by training policies and Q-functions that take as an additional input a goal $g \in \mathcal{G}$ [41, 3], e.g. $a_t = \pi(s_t, g)$. A universal policy can hence be trained by using arbitrary RL algorithms.

Following UVFA [41], the sparse reward formulation $r(s_t, a, g) = [d(s_t, g) < \epsilon]$ will be used in this work, where the agent gets a positive reward when the distance $d(.,.)$ between the current state and the goal is less than ϵ. In the context of a robot performing the task of picking and placing an object, this means that the robot gets a reward only if the object is within ϵ Euclidean distance of the desired goal location of the object. Having a sparse reward overcomes the limitation of hand engineering the reward function, which often requires extensive domain knowledge. However, sparse rewards are not very informative and makes it hard to optimize. In order to overcome the difficulties with sparse rewards, we employ a recent method: Hindsight Experience Replay (HER) [3].

D. Hindsight Experience Replay (HER)

HER [3] is a simple method of manipulating the replay buffer used in off-policy RL algorithms that allows it to learn universal policies more efficiently with sparse rewards. After experiencing some episode $s_0, s_1, ..., s_T$, every transition $s_t \rightarrow s_{t+1}$ along with the goal for this episode is usually stored in the replay buffer. However with HER, the experienced transitions are also stored in the replay buffer with different goals. These additional goals are states that were achieved later in the episode. Since the goal being pursued does not influence the environment dynamics, we can replay each trajectory using arbitrary goals assuming we use an off-policy RL algorithm to optimize [35].

IV. METHOD

We now describe our method along with the technique of bottlenecks to speed up training. Following this, we also describe domain randomization for transferring simulator learned policies to the real robot.

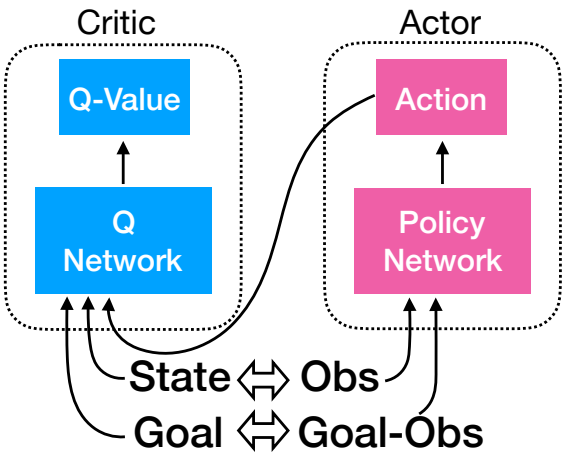

Fig. 2. Having asymmetric inputs, i.e. full states for the critic and partial observations for the actor improves training. In the multi goal setting, the critic additionally requires full goal states while the actor additionally requires partial observations for the goal.

A. Asymmetric Actor Critic

In its essence our method builds on *actor-critic* algorithms [19] by using the full state $s_t \in \mathcal{S}$ to train the critic, while using partial observation $o_t \in \mathcal{O}$ to train the actor (see Figure 2). Note that s_t is the underlying full state for the observation o_t. In our experiments, observations o_t are images taken by an external camera.

Algorithm 1 Asymmetric Actor Critic

Initialize actor-critic algorithm $\mathcal{A}$
Initialize replay buffer R
for episode= $1, M$ **do**
  Sample a goal g and an initial state s_0
  Render goal observation g^o
    $g^o \leftarrow \texttt{render}(g)$
  for $t = 0, T-1$ **do**
    Render image observation o_t
      $o_t \leftarrow \texttt{render}(s_t)$
    Obtain action a_t using behavioural policy:
      $a_t \leftarrow \pi_b(o_t, g^o)$
    Execute action a_t, receive reward r_t and transition to s_{t+1}
    Store $(s_t, o_t, a_t, r_t, s_{t+1}, o_{t+1}, g, g^o)$ in R
  end for
  for n=1, N **do**
    Sample minibatch $\{s, o, a, r, s^{'}, o^{'}, g, g^o\}_0^B$ from R
    Optimize critic using $\{s, a, r, s^{'}, g\}_0^B$ with $\mathcal{A}$
    Optimize actor using $\{o, a, r, o^{'}, g^o\}_0^B$ with $\mathcal{A}$
  end for
end for

The algorithm (described in Algorithm 1), begins with initializing the networks for an off-policy actor-critic algorithm $\mathcal{A}$ [35]. In this paper, we use DDPG [23] as the actor-critic algorithm. The replay buffer R used by this algorithm is initialized with no data. For each episode, a goal g and an initial state s_0 are sampled before the rollout begins. g^o is the rendered goal observation. At every timestep t of the episode, a partially observable image of the scene o_t is rendered from the simulator at the full state s_t. The behavioural policy from $\mathcal{A}$, which is usually a noisy version of the actor is used to generate the action a_t for the agent/robot to take. After taking this action, the environment transitions to the next state s_{t+1}, with its corresponding rendered image o_{t+1}.

Since DDPG relies on a replay buffer to sample training data, we build the replay buffer from the episodic experience $(s_t, o_t, a_t, r_t, s_{t+1}, o_{t+1}, g, g^o)$. To improve performance for the sparse reward case, we augment the standard replay buffer by adding hindsight experiences [3].

After the episodic experience has been added to the replay buffer R, we can now train our actor-critic algorithm $\mathcal{A}$ from a sampled minibatch of size B from R. This minibatch can be represented as $\{s, o, a, r, s^{'}, o^{'}, g, g^o\}_0^B$, where $s^{'}$ and $o^{'}$ are the next step full state and next step observation respectively. Since the critic takes full states as input, it is trained on $\{s, a, r, s^{'}, g\}_0^B$. Since the actor takes observations as input, it is trained on $\{o, a, r, o^{'}, g^o\}_0^B$. We experimentally show that asymmetric inputs for the critic and actor significantly improves performance and supports transfer of more complex manipulation behaviours to real robots.

B. Improvements with bottlenecks

One way of improving the efficiency of training is to use bottlenecks [52]. The key idea is to constrain one of the actor network's intermediary layers to predict the full state. Since the full state is often of a smaller dimension than the other layers of the network, this state predictive layer is called the bottleneck layer.

C. Randomization for transfer

A powerful technique for domain transfer of policies from rendered images to real world images is domain randomization [47, 40]. The key idea is to randomize visual elements in the scene during the rendering. Learning policies with this randomization allows the policy to generalize to sources of error in the real world and latch on to the important aspects of the observation.

For the purposes of this paper, we randomize the following aspects: texture, lighting, camera location and depth. For textures, random textures are chosen among random RGB values, gradient textures and checker patterns. These random textures are applied on the different physical objects in the scene, like the robot and the table. For lighting randomization, we randomly switch on lighting sources in the scene and also randomize the position, orientation and the specular characteristics of the light. For camera location, we randomize the location of the monocular camera in a box around the expected location of the real world camera. Furthermore, we randomize the orientation and focal length of the camera and add uniform noise to the depth. RGB samples of randomization on the *Fetch Pick* environment can be seen in Figure 3.

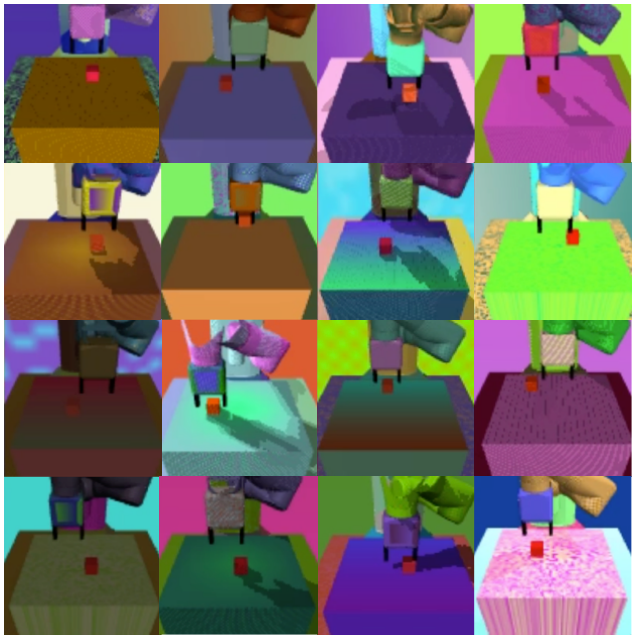

Fig. 3. To enable transfer of policies from the simulator to the real world, we randomize various aspects of the renderer during training. These aspects include textures, lighting and the position of the camera.

V. RESULTS

To show the effectiveness of our method, we experiment on a range of simulated and real robot environments. In this section we first describe the environments. Following this, we discuss comparisons of our methods to baselines and show the utility of our method for improving training. Finally, we discuss real robot experiments.

A. Environments

Since there are no standard environments for multi-goal RL, we create three of our own simulated environments to test our method. The first two environments, *Particle* and *Reacher* are in a 2D workspace. The third environment *Fetch Pick* is in a 3D workspace with a simulated version of the Fetch robot picking up and placing a block. All these environments are simulated in the *MuJoCo* [48] physics simulator.

(a) Particle: In this 2D environment the goal for the agent is to move a 2D particle to a given location. The state space is 2D and consists of the particle's location. The observation space is RGB images ($100 \times 100 \times 3$) from a camera placed above the scene. The action space is the 2D velocity of the particle. This action space allows for control on single RGB observations without requiring memory for velocity (since velocity cannot be inferred from a single RGB frame). The agent gets a sparse reward ($+1$) if the particle is within ϵ of the desired goal position and no reward (0) otherwise. The observation for the goal is an image of the particle in its desired goal position.

(b) Reacher: In this 2D environment the goal for the agent is to move the end-effector of a two-link robot arm to a target location. The state space is 2D and consists of the joint positions. The observation space is RGB images ($100 \times 100 \times 3$) from a camera placed above the scene. The action space is the 2D velocities for the joints. The agent gets a sparse reward ($+1$) if the end-effector is within ϵ of the desired goal position and no reward (0) otherwise. The observation for the goal is an image of the reacher in its desired goal end-effector position.

(c) Fetch Pick: In this 3D environment with the simulated Fetch robot, the goal for the agent is to pick up the block on the table and move it to a given location in the air. The state space consists of the joint positions of the robot and the location of the block on the table. The observation space is RGBD images ($100 \times 100 \times 4$) from a camera placed in front of the robot. The action space is 4D. Since this problem does not require gripper rotation, we keep it fixed. Three of the four dimensions of the action space specify the desired relative[1] position for the gripper. The last dimension specifies the desired distance between the fingers of the gripper. The agent gets a sparse reward ($+1$) if the block is within ϵ of the desired goal block position. The observation for the goal is an image of the block in its desired goal block position and the Fetch arm in a random position. To make exploration in this task easier following [3], we record a *single* state in which the box is grasped and start half of the training episodes from this state.

B. Robot evaluation

For our real world experiments we use a 7-DOF Fetch robotic arm[2], which is equipped with a two fingered parallel gripper. The camera observations for the real world experiments is an off the shelf Intel RealSense R200 camera that can provide aligned RGBD images. Since real depth often contains holes [27], we employ nearest neighbour hole filling to get better depth images [50]. To further improve the depth, we cover/recolor parts of the robot that are black like parts of the torso and parts of the gripper.

We experiment on three tasks for the real robot. The first task is *Pick* which is similar to the simulated task of *Fetch Pick* described in Section V-A(c). The second task is *Forward Push*, where the robot needs to push a block forward[3]. The third task is *Block Move*, where the robot needs to move the block to the target position on the table. In all tasks the goal is specified by an image of the block in the target location. The manipulation behaviour is considered to be successful if the block is within $2cm$ of the desired target location. A video of these experiments can be found in the supplementary video and sample successes from our method in Figure 7.

The observations for the real robot tasks is an RGBD image from the physical camera placed in front of the robot. The goal observation for the actor is a simulated image describing the desired goal. We note that giving real world observations for

[1]The desired gripper position is relative to the current gripper position.
[2]fetchrobotics.com/platforms-research-development/
[3]The fingers are blocked for this task to avoid grasping.

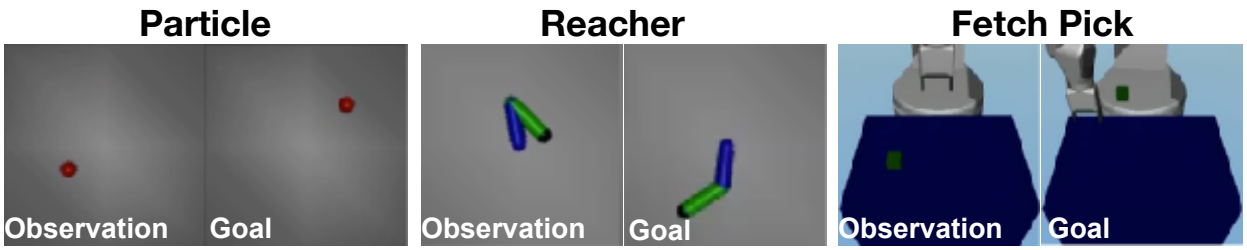

Fig. 4. To evaluate our method, we test on three different environments: *Particle*, *Reacher* and *Fetch Pick*. Since we learn multi goal policies, the policy takes in both the observation at timestep t and the desired goal for the episode.

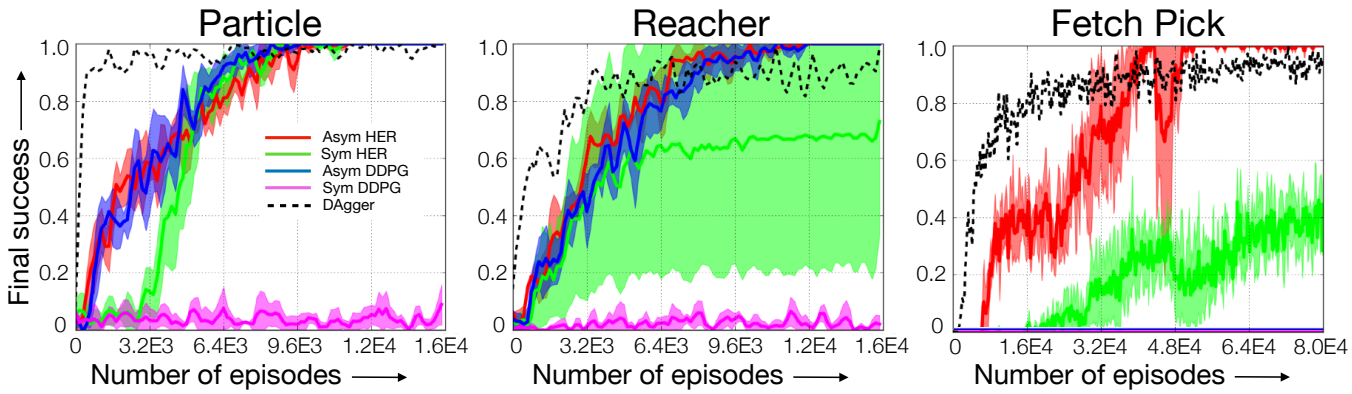

Fig. 5. We show that asymmetric inputs for training outperforms symmetric inputs by significant margins on our simulated environments. The shaded region corresponds to ± 1 standard deviation across 5 random seeds. The y-axis corresponds to the sparse reward obtained at the end of an episode averaged over 100 runs. Although the behaviour cloning (BC) by expert imitation baseline (dashed lines) learns faster initially, it saturates to a sub optimal value compared to asymmetric HER. Also note that the BC baseline doesn't include the iterations the expert policy was trained on.

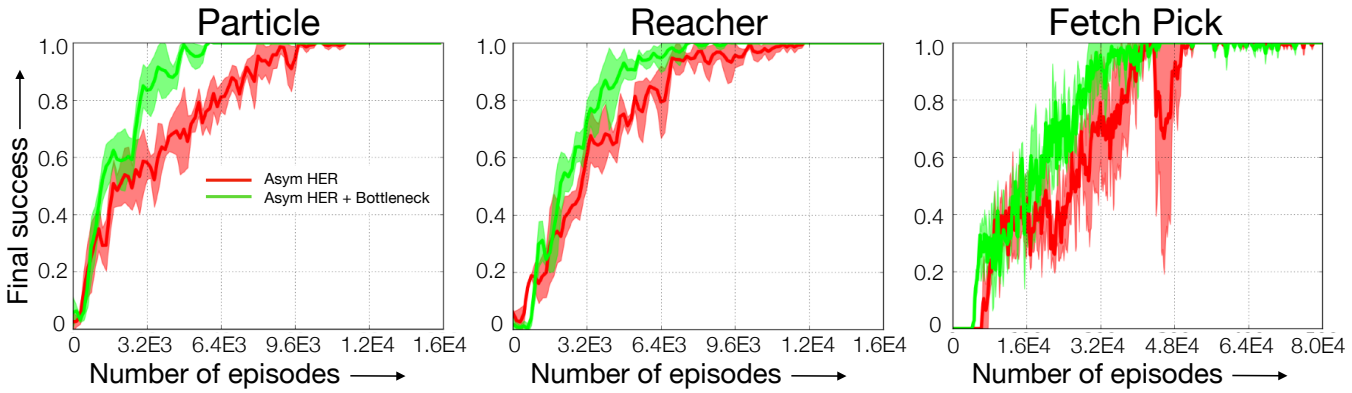

Fig. 6. We show that bottlenecks can be used to further improve training of our method. The shaded region corresponds to ± 1 standard deviation across 5 random seeds. The y-axis corresponds to the sparse reward obtained at the end of an episode averaged over 100 runs. On the *Particle* and *Reacher*, the improvements are quite significant. On *Fetch Pick*, we observe more stable training (lower variance denoted by shaded regions).

the goal observation also works, however for consistency in evaluation, we use a simulated goal observation.

C. Do asymmetric inputs to the actor critic help?

To study the effect of asymmetric inputs, we compare to the baseline of using symmetric inputs (images for both the actor and the critic networks). Figure 5 shows a summary of the final episodic rewards, with the x-axis being the number of episodes the agent experiences. As evident from the *Particle* results, asymmetric input versions of both DDPG and HER perform much better than their symmetric counterparts. The simplicity of the *Particle* may explain the similar performance between asymmetric DDPG and HER. *Fetch Pick* is a much harder sparse reward task, which shows the importance of using HER over DDPG. In this case as well, the asymmetric version of HER performs significantly better than the symmetric version. This experiment hence demonstrates the significant utility of asymmetric training for learning visual policies.

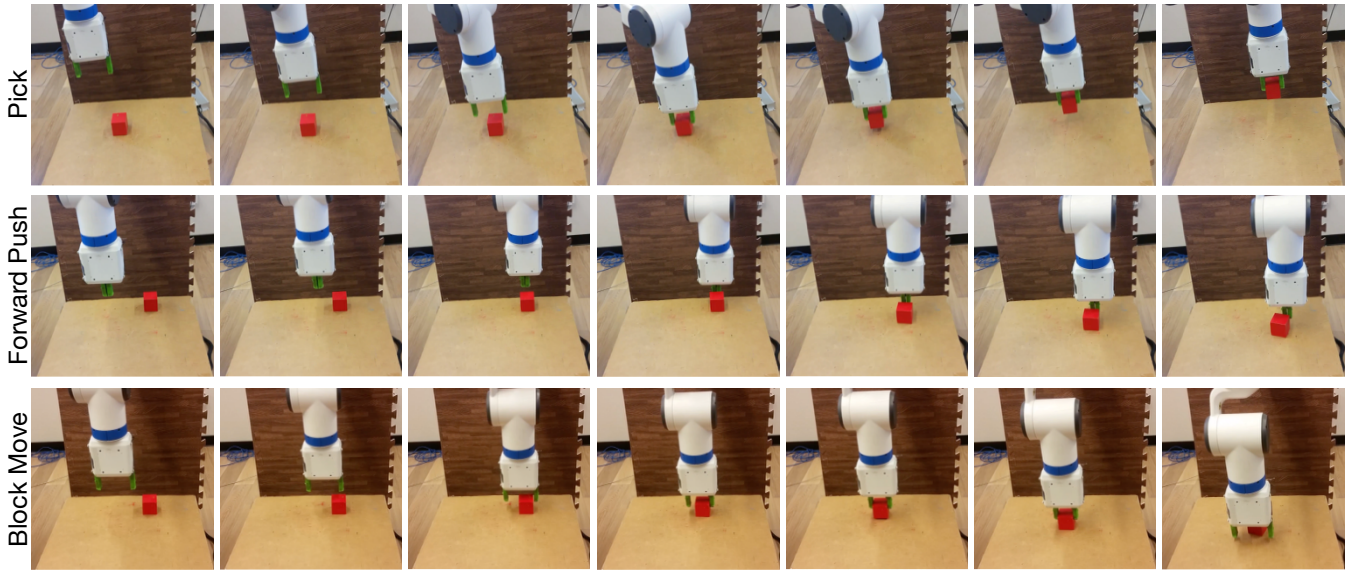

Fig. 7. Successive frames of our asymmetric HER policies on three real robot tasks show how our method can be successfully used for simulation to real transfer of complex policies. Full length experiments can be found in the supplementary video.

TABLE I
COMPARISON OF ASYMMETRIC HER WITH BASELINES AND ABLATIONS

	Asym HER	Baselines				Randomization ablations	
		Asym DDPG	Sym HER	Vanilla BC	DAgger BC	Without any	Without viewpoints
Pick	**5/5**	0/5	0/5	0/5	3/5	0/5	0/5
Forward Push	**5/5**	0/5	0/5	1/5	0/5	0/5	0/5
Block Move	**5/5**	0/5	0/5	0/5	0/5	0/5	4/5

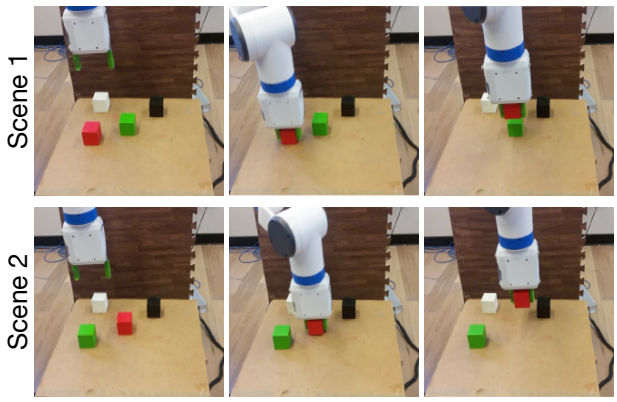

Fig. 8. Domain randomization during training allows the learned policies to be robust to distractors. Here we see how a policy trained to *Pick* the red block is robust to distractor blocks. The difference in the two scenes shown here is that inspite of changing the initial location of the red block, the arm still picks the red block.

D. Would state-based learning followed by visual behavioral cloning succeed?

Imitation learning is a powerful technique in robotics [4]. Hence a much stronger baseline is to behaviour clone from an expert policy. To do this we first train an expert policy [4] on full states that performs the task perfectly. Now given this expert policy, we behaviour clone to a policy that takes the visual observations as input [15]. Furthermore, since we have access to the expert state-based policy, we can improve the imitated visual policy using DAgger [38].

Figure 5 shows the final episodic rewards of the behaviour cloned policy (in dashed lines) with the x-axis being the number of demonstrations. As expected, the DAgger policy learns much faster initially (since it receives supervision from a much stronger expert policy). However in all the environments, it saturates in performance and is lower than than our method (asymmetric HER) for a large number of rollouts. One reason for this is that behaviour cloning would fail if the expert policy depends on information contained in the full state but not in the partial observation. This experiment shows that although behaviour cloning can help speed up visual learning, it cannot overcome the domain mismatch between the expert and cloned policy. These findings are consistent with the results in James et al. [15]. Asymmetric Actor Critic does not face this challenge since state information is used only in the training of the critic.

E. Does the bottleneck layer speed up training?

Another way of incorporating the full state from the simulator is by adding an auxiliary task of predicting the full state from partial observations. By adding a bottleneck layer [52] in the actor and adding an additional L_2 loss between the

bottleneck output and the full state, we further speed up training. On our simulated tasks, these bottlenecks in the policy network improve the stability and speed of training (as seen in Figure 6).

F. How well do these policies transfer to a real robot?

By combining asymmetric HER with domain randomization [47], we show significant performance gains (see Table I) compared to baselines previously mentioned. Note that we measure success if the block is within 2*cm* of the desired target position. Our method succeeds on all the three tasks for all the 5 times the policy were run with different block initializations and goals. We also note that behaviours like *push-grasping* [9] and *re-grasping* [42] emerge from these trained policies which can be seen in the www.goo.gl/b57WTs. Among the baselines we evaluate against, we note that behaviour cloning with DAgger is the only one that performs reasonably (as seen in the video and Table I).

This experiment demonstrates the utility of our method on robots with minimal instrumentation. Using RGBD inputs in this fashion means that error-prone aspects like state estimation or precise camera calibration aren't required. Moreover, since the goal is defined by an image, this can be given by showing the camera the desired scene. Alternatively, if it is easier to describe the goal by the state of the block, a rendered image of the block in the desired state can be used as the goal.

G. How important is domain randomization?

To show the importance of randomization, we perform ablations (last two columns of Table I) by training policies without any randomization and testing them in the real world. We notice that without any randomization, the policies fail to perform on the real robot while performing perfectly in the simulator. Another randomization ablation is by removing the viewpoint randomization while keeping the texture and lighting randomization during training. We notice that apart from the *Block Move* task, removing viewpoint randomization severely affects performance.

Randomizing the observations in training also gives us an added benefit: robustness to distractors. Figure 8 shows the performance of our *Pick* policy, which was trained on a single red block, work even in the presence of distractor blocks.

H. Implementation Details

In this section we provide more details on our training setup. The critic is a fully connected neural network with 3 hidden layers; each with 512 hidden units. The hidden layers use ReLU [20] for the non linear activation. The input to the critic is the concatenation of the current state s_t, the desired goal state g and the current action a_t. The actor is a convolutional neural network (CNN) with 4 convolutional layers with 64 filters each and kernel size of 2×2. This network is applied to both the current observation o_t and the goal observation g^o. The outputs of both CNNs are then concatenated and passed through a fully connected neural network with 3 hidden layers. Similar to the critic, the hidden layers have 512 hidden units each with ReLU activation. The output of this actor network is normalized by a tanh activation and rescaled to match the limits of the environment's action space. In order to prevent tanh saturation, we penalize the preactivations in the actor's cost.

During each iteration of DDPG, we sample 16 parallel rollouts of the actor. Following this we perform 40 optimization steps on minibatches of size 128 from the replay buffer of size 10^5 transitions. The target actor and critic networks are updated every iteration with a Polyak averaging of 0.98. We use Adam [17] optimization with a learning rate of 0.001 and the default Tensorflow [1] values for the other hyperparameters. We use a discount factor of $\gamma = 0.98$ and use a fixed horizon of $T = 50$ steps. For efficient learning, we also normalize the input states by running averages of the means and standard deviations of encountered states.

The behavioural policy chosen for exploration chooses a uniform random action from the space of valid actions with probability 20%. For the rest 80% probability, the output of the actor is added with coordinate independent Normal noise with standard deviation equal to 5% of the action range.

VI. CONCLUSION

In this work we introduce asymmetric actor-critic, a powerful way of utilizing the full state observability in a simulator. By training the critic on full states while training its actor on rendered images, we learn vision-based policies for complex manipulation tasks. Extensive evaluation both in the simulator and on our real world robot shows significant improvements over standard actor-critic baselines. This method's performance is also superior to the much stronger imitation learning with DAgger baseline, even though it was trained without an expert. Coupled with domain randomization, our method is able to learn visual policies that works in the real world while being trained solely in a simulator.

REFERENCES

[1] Martín Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *NIPS*, 2017.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 2009.

[5] Zdravko Balorda. Reducing uncertainty of objects by robot pushing. In *ICRA 1990*.

[6] A Bicchi and V Kumar. Robotic grasping and contact: a review. In *ICRA 2000*.

[7] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[8] Mark Cutler and Jonathan P How. Efficient reinforcement learning for robots using informative simulated priors. In *ICRA 2015*.

[9] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. *Robotics: Science and Systems (RSS)*, 2011.

[10] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[11] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[12] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *arXiv preprint arXiv:1703.00727*, 2017.

[13] David Held, Zoe McCarthy, Michael Zhang, Fred Shentu, and Pieter Abbeel. Probabilistically safe policy transfer. *ICRA 2017*.

[14] Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*, 2016.

[15] Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

[16] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 1996.

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] J Zico Kolter and Andrew Y Ng. Learning omnidirectional path following using dimensionality reduction. In *RSS*, 2007.

[19] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS 2012*.

[21] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR 2016*.

[22] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *ISER*, 2016.

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

[25] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[26] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[27] Kwan-Jung Oh, Sehoon Yea, and Yo-Sung Ho. Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video. In *PCS 2009*.

[28] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *ICRA 2016*, .

[29] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *ICRA 2017*, .

[30] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016.

[31] Lerrel Pinto, James Davidson, and Abhinav Gupta. Supervision via competition: Robot adversaries for learning tasks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1601–1608. IEEE, 2017.

[32] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. *ICML*, 2017.

[33] Benjamin Planche, Ziyan Wu, Kai Ma, Shanhui Sun, Stefan Kluckner, Terrence Chen, Andreas Hutter, Sergey Zakharov, Harald Kosch, and Jan Ernst. Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition. *arXiv preprint arXiv:1702.08558*, 2017.

[34] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992.

[35] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, 2001.

[36] Aravind Rajeswaran, Sarvjeet Ghotra, Sergey Levine, and Balaraman Ravindran. Epopt: Learning robust neural network policies using model ensembles. *ICLR*, 2017.

[37] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV 2016*.

[38] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction

to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[39] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[40] Fereshteh Sadeghi and Sergey Levine. (cad)2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[41] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML 2015*.

[42] Thomas Schlegl, Martin Buss, Toru Omata, and Günther Schmidt. Fast dextrous re-grasping with optimal contact forces and contact sensor-based impedance control. In *ICRA 2001*.

[43] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML 2015*.

[44] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML 2014*.

[45] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[46] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[47] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *IROS*, 2017.

[48] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[49] Ulrich Viereck, Andreas ten Pas, Kate Saenko, and Robert Platt. Learning a visuomotor controller for real world robotic grasping using easily simulated depth images. *arXiv preprint arXiv:1706.04652*, 2017.

[50] Na-Eun Yang, Yong-Gon Kim, and Rae-Hong Park. Depth hole filling using the depth distribution of neighboring regions of depth holes in the kinect sensor. In *ICSPCC 2012*.

[51] Wenhao Yu, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[52] Fangyi Zhang, Jürgen Leitner, Ben Upcroft, and Peter Corke. Vision-based reaching using modular deep networks: from simulation to the real world. *arXiv preprint arXiv:1610.06781*, 2016.