Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Florian Golemo
INRIA Bordeaux & MILA
florian.golemo@inria.fr

Adrien Ali Taïga
MILA, Université de Montréal
adrien.ali.taiga@umontreal.ca

Pierre-Yves Oudeyer
INRIA Bordeaux
pierre-yves.oudeyer@inria.fr

Aaron Courville [*]
MILA, Université de Montréal
aaron.courville@umontreal.ca

Abstract: Despite the recent successes of deep reinforcement learning, teaching complex motor skills to a physical robot remains a hard problem. While learning directly on a real system is usually impractical, doing so in simulation has proven to be fast and safe. Nevertheless, because of the "reality gap," policies trained in simulation often perform poorly when deployed on a real system. In this work, we introduce a method for training a recurrent neural network on the differences between simulated and real robot trajectories and then using this model to augment the simulator. This Neural-Augmented Simulation (NAS) can be used to learn control policies that transfer significantly better to real environments than policies learned on existing simulators. We demonstrate the potential of our approach through a set of experiments on the Mujoco simulator with added backlash and the Poppy Ergo Jr robot. NAS allows us to learn policies that are competitive with ones that would have been learned directly on the real robot.

1 Introduction

Reinforcement learning (RL) with function approximation has demonstrated remarkable performance in recent years. Prominent examples include playing Atari games from raw pixels [1], learning complex policies for continuous control [2, 3], and surpassing human performance on the game of Go [4]. However, most of these successes were achieved in non-physical environments: simulations, video games, etc. Learning complex policies on physical systems remains an open challenge. Typical reinforcement learning methods require a large amount of interaction with the environment and, in addition, it is also often impractical, if not dangerous, to roll out a partially trained policy. Both these issues make it unsuitable for many tasks to be trained directly on a physical robot.

The fidelity of current simulators and rendering engines provides an incentive to use them as training grounds for control policies, hoping the newly acquired skills would transfer back to reality. However, this is not as easy as one might hope since no simulator perfectly captures reality. This problem is widely known as the *reality gap*. See Figure 1 for an illustration.

The reality gap can be seen as an instance of a *domain adaptation* problem, where the input distribution of a model changes between training (in simulation) and testing (in the real world). In the case of image classification and off-policy image-based deep reinforcement learning, this issue has sometimes been tackled by refining images from the *source domain* – where the training happens – to appear closer to images from the *target domain* – where evaluation is carried out [5, 6].

In this work, we take a similar approach and apply it to continuous control. We use data collected from a real robot to train a recurrent neural network to predict the discrepancies between simulation and the real world. This allows us to improve the quality of our simulation by grounding it in realistic trajectories. We refer to our approach as *Neural-Augmented Simulation* (NAS). We can use it with any reinforcement learning algorithm, combining the benefits of simulation and fast offline training,

[*] CIFAR Fellow

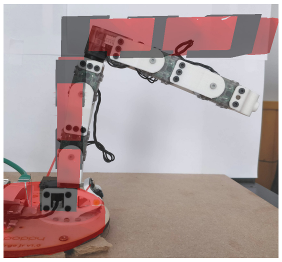


Figure 1: Impact of backlash: when setting both the simulated robot (red) and the real robot (white) to the resting position, the small backlash of each joint adds up to a noticeable difference in the end effector position.

while still learning policies that transfer well to robots in real environments. Since we collect data using a non-task-specific policy, NAS can be used to learn policies related to different tasks. Thus, we believe that NAS provides an efficient and effective strategy for multi-task sim-to-real transfer.

With our NAS strategy, we aim to achieve the best of both modeling modalities. While the recurrent neural network compensates for the unrealistically modeled aspects of the simulator; the simulator allows for better extrapolation to dynamic regimes that were not well explored under the data collection policy. Our choice to use a recurrent model[2] is motivated by the desire to capture deviations that could violate the standard Markov assumption on the state space.

We evaluate our approach on two OpenAI Gym [8] based simulated environments with an artificially created reality gap in the form of added backlash. We also evaluate on the Poppy Ergo Jr robot arm, a relatively inexpensive arm with dynamics that are not well modeled by the existing simulator. We find that NAS provides an efficient way to learn policies that approach the performance of policies trained directly on the physical system.

2 Related work

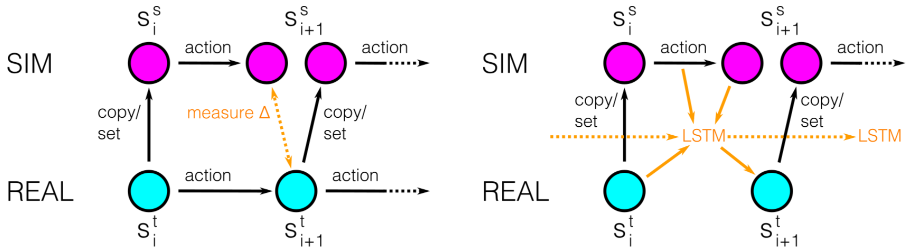
Despite the recent success of deep reinforcement learning, learning on a real robot remains impractical due to poor sample efficiency. Classical methods for sim-to-real transfer include co-evolving the policy and the simulation [9, 10], selecting for policies that transfer well [11, 12], learning a transfer function of policies [13], and modeling different levels of noise [14]. Nonetheless, these methods are usually limited to domains with a low-dimensional state-action space.

An alternative is to try to learn a forward model of the dynamics, a mapping between the current state and an action to the next state [15, 16, 17]. However, despite some successes, learning a forward model of a real robot remains an open challenge. The compounding error of these models quickly deteriorates over time and corrections are needed to compensate for the uncertainty of the model [18]. These methods are also usually sensitive to the data used to train the model, which is often different from the state distribution encountered during policy learning [19, 20]

Domain adaptation has received a lot of focus from the computer vision community. There have been many approaches such as fine-tuning a model trained on the source domain using data from the target domain [21], enforcing invariance in features between data from source and target domain [22] or learning a mapping between source and target domain [23].

Similar ideas have been applied in RLx, such as previous work that focused on learning internal representations robust to change in the input distribution [24]. Using data generated from the simulation during training was also used in bootstrapping the target policy [25], Progressive Neural Networks [26] were also used to extend a simulation-learned policy to different target environments, but the most common method remains learning the control policy on the simulator before fine tuning on the real robot. Another quite successful recent method consists of randomizing all task-related physical properties of the simulation [27, 28]. This approach is very time-consuming because in order for a policy to generalize to all kinds of inputs, it has to experience a very wide range of noise during learning.

[2] We chose to use a Long Short-Term Model (LSTM) [7] as our recurrent neural network.



(a) Training phase

(b) Policy Learning Phase

Figure 2: Left: Overview of the method for training the forward dynamics model. By gathering state differences when running the same action in simulation and on the real robot. Right: When the forward model is learned, it can be applied to simulator states to get the corresponding real state. This correction model (Δ) is time-dependent (implemented as LSTM). The policy learning algorithm only has access to the "real" states.

Recently an approach very similar to our has been developed in the work of [29]. However, rather than learning a state space transition, the authors learn a transformation of actions. This learning is done on-policy so once the model is learned, the simulator augmentation doesn't transfer to other tasks.

In summary, existing approaches struggle with either computational complexity or with difficulties in transferability to other tasks. We would like to introduce a technique that is capable of addressing both these issues.

3 Method

3.1 Notation

We consider a domain adaptation problem between a source domain D_s (usually a simulator) and a target domain D_t (typically a robot in the real world). Each domain is represented by a Markov Decision Process (MDP) $\langle S, A, p, r, \gamma \rangle$, where S is the state space, A the action space, for $s \in S$ and $a \in S$, $p(.|s, a)$ the transition probability of the environment, r the reward and γ the discount factor. We assume that the two domains share the same action space, however, because of the perceptual-reality gap state space, dynamics and rewards can be different even if they are structurally similar, we write $D_s = \langle S_s, A, p_s, r_s \rangle$ and $D_t = \langle S_t, A, p_t, r_t \rangle$ for the two MDPs. We also assume that rewards are similar (i.e $r_s = r_t$) and, most importantly, we assume that we are given access to the source domain and can reset it to a specific state. For example, for transfer from simulation to the robot, we assume we are free to reset the joint positions and angular velocities of the simulated robots.

3.2 Target domain data collection

We model the difference between the source and target domain, learning a mapping from trajectories in the source domain to trajectories in the target domain. Given a behavior policy μ (which could be random or provided by an expert) we collect trajectories from the target environment (i.e. real robot) follow actions given by μ.

3.3 Training the model

To train the LSTM in NAS, we sample an initial state $s_0 \sim p_t(s_0)$ from the target domain distribution and set the source domain to start from the same initial state (i.e $s_0^s = s_0^t = s_0$). At each time step an action is sampled following the behavior policy $a_i \sim \mu(s_i^t)$, then executed on the two domains to get the transition $(s_i, a_i, s_{i+1}^s, s_{i+1}^t)$. The source domain is then reset to the target domain state and the procedure is repeated until the end of the episode. The resulting trajectory $\tau = (s_0, a_0, s_1^s, s_1^t, ..., s_{T-2}^s, a_T, s_{T-1}^s, s_{T-1}^t)$ of length T is then stored, the procedure is described in Algorithm 1.

Algorithm 1 Offline Data Collection

```
Initialize model $\phi$
for episode= $1, M$ do
    Initialize the target domain from a state $s_1^t$
    Initialize the source domain to the same state $s_1^s = s_1^t$
    for i = 1, T do
        Select action $a_i \sim \mu(s_i^t)$ according to the behavior policy
        Execute action $a_i$ on the source domain and observe new state $s_{i+1}^s$
        Execute action $a_i$ on the target domain and observe new state $s_{i+1}^t$
        Update $\phi$ with the tuple $s_{i+1}^s = s_{i+1}^t$
        Set $s_{i+1}^s = s_{i+1}^t$
    end for
end for
```

Algorithm 2 Policy Learning in Source Domain

```
Given a RL algorithm $\mathbb{A}$
A model $\phi$
Initialize $\mathbb{A}$
for episode= $1, M$ do
    Initialize the source domain from a state $s_1^s$
    Initialize the $\phi$ latent variables $h$
    for i = 1, T do
        Sample action $a_i \sim \pi(s_i^s)$ using the behavioral policy from $\mathbb{A}$
        Execute action $a_i$ in the source domain, observe a reward $r_i$ and a new state $s_{i+1}^s$
        Set $\hat{s}_{i+1}^t = s_{i+1}^s + \phi(s_i, a_i, h, s_{i+1}^s)$ the estimate of $s_{i+1}^t$ given by $\phi$
        Do a one-step policy update with $\mathbb{A}$ using the transition $(s_i^s, a_i, \hat{s}_{i+1}^t, r_i)$
        Set the source domain to $s_{i+1}^s = \hat{s}_{i+1}^t$
    end for
end for
```

After collecting the data the model ϕ, an LSTM [7], is trained to predict s_{i+1}^t. The difference between two states is usually small, so the network outputs a correction term $\phi(s_i^t, a_i, h, s_{i+1}^s) = s_{i+1}^t - s_{i+1}^s$ where h is the hidden state of the network. We also compare our approach with a forward model trained without using information from the source domain, $\psi(s_i, a_i, h) = s_{i+1}^t - s_i$. We normalize the inputs and outputs, and the model is trained with maximum likelihood using Adam [30].

3.4 Transferring to the target domain

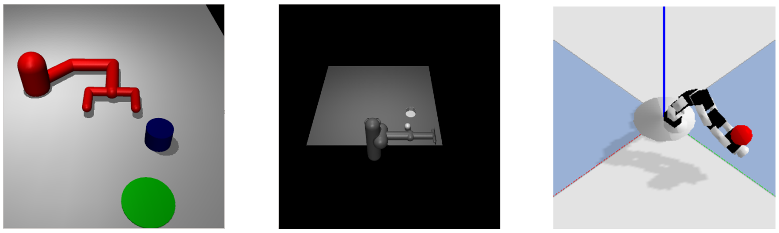
Once the model ϕ is trained, it can be combined with the source environment to learn a policy that will later be transferred to the target environment. We use PPO[3] [31] in all our experiments but any other reinforcement learning algorithm could be used. During learning at each time step the current state transition in the source domain is passed to ϕ to compute an estimate of the current state in the target environment, an action a_i is chosen according to this estimate $a_i \sim \pi(\phi(s_i^s, s_{i+1}^s))$, then the source domain state is set to the current estimate of the target domain state $\sim \phi(s_i^s, s_{i+1}^s)$. This will allow our model to correct trajectories from the source domain, making them closer to the corresponding trajectory in the target domain.

4 Experiments

We evaluated our method on a set of simulated robotic control tasks using the MuJoCo physics engine [32] as well as on the open-source 3D-printed physical robot "Poppy Ergo Jr" [33] (see `https://www.poppy-project.org/en/robots/poppy-ergo-jr`)[4].

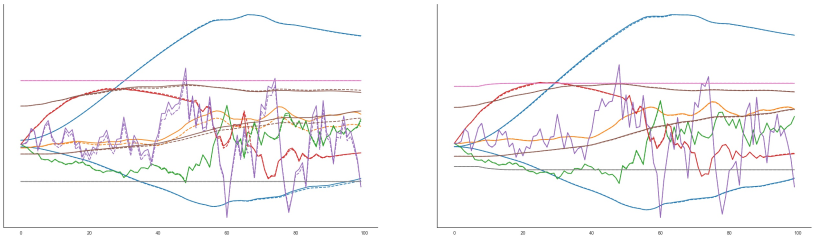
[3] A list of all our hyperparameters is included in the supplementary material.

[4] The code for our experiments can be found at `https://github.com/aalitaiga/sim-to-real/`



(a) Pusher (b) Striker (c) ErgoReacher

Figure 3: Benchmark simulation environments used



(a) Forward model rollout (b) Source environment + model rollout

Figure 4: Starting from the same initial state, a fixed sequence of actions is executed in D_t and compared with the trajectory derived from a forward model and our method. The solid line depicts the true trajectory in D_t whereas the dotted line represents the trajectory using our corrected model, 4a is just an LSTM and 4b is an LSTM and model ψ

4.1 Simulated Environments Overview

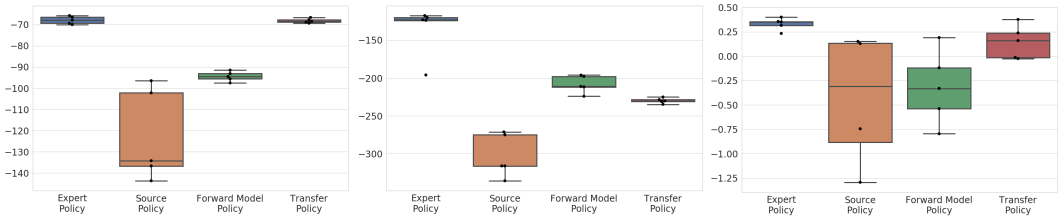
We created an artificial reality gap using two simulated environments. The source and target environments are identical, except that the target environment has backlash. We also experimented with small changes in joints and link parameters but noted that it did not impact our results. Overall, the difference in backlash between the environments was significant enough to prevent policies trained on the source environment from performing well on the target environment (though an agent could solve the target environment if trained on it). More details about why we picked backlash can be found in appendix A.2.

We used the following environments from OpenAI Gym for our experiments:

- *Pusher*: A 3-DOF arm trying to move a cylinder on a plane to a specific location.
- *Striker*: A 7-DOF arm that has to strike a ball on a table in order to make the ball move into a goal that is out of reach for the robot.
- *ErgoReacher*: A 4-DOF arm that has to reach a goal spot with its end effector.

While we added backlash to only one joint of the Pusher, to test the limits of our artificial reality gap, we added backlash to three joints of the Striker and two joints or ErgoReacher. We also compare our proposed method with different baselines:

- *Expert policy*: policy trained directly in the target environment
- *Source policy*: transferring a policy trained in source environment without any adaption
- *Forward model policy*: a forward dynamic model ψ is trained using an LSTM and data collected from the target domain then a policy trained using only this model (without insight from the source domain)
- *Transfer policy*: the policy trained using NAS



(a) Pusher (b) Striker (c) ErgoReacher

Figure 5: Comparison of the different methods described when deployed on the target environment, for the Pusher (5a) and the Striker (5b).

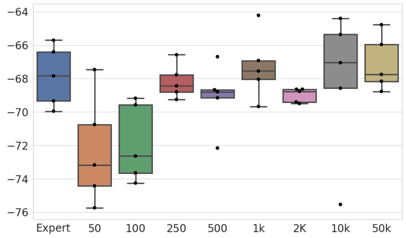


Figure 6: We compare the results of our method when the number of trajectories used to train the model ϕ varies on the Pusher

4.2 Trajectory following

We evaluated the two models learned ϕ and ψ on a 100-step trajectory rollout on the Pusher (see Figure 4). The forward model ψ is trained without access to the source domain and is making predictions in an open-loop. While this model is accurate at the beginning of the rollout, its small error compounds over time. We will see later that this makes it hard to learn policies that will achieve a high reward. On the other hand, the model ϕ is grounded using the source environment and only needs to correct the simulator predictions, so the difference between the trajectory in the real environment is barely noticeable. It shows that correcting trajectories from the source environment provide an efficient way to model the target environment.

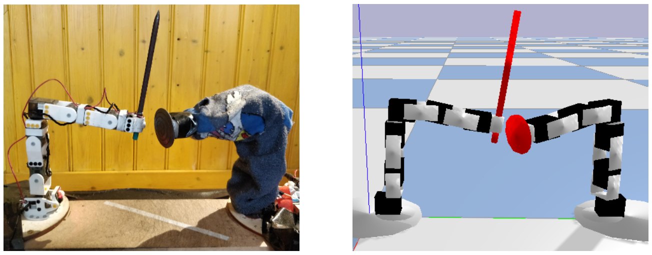
4.3 Simulated Environments - Sim to Sim transfer

We tested our method on the simulated environments previously introduced. The number of trajectories used to train the models varied from 250 for the Pusher over 1k for the ErgoReacher to 5k for the Striker. Policies are trained for 2M frames and evaluated over 100 episodes, averaged across 5 seeds, results are given in Figure 5. Additional information about the training can be found in app. A.1.

Our experiments show that in all our environments the policy learned using NAS improve significantly over the source policy, and even reach expert performance on the Pusher. Though it seems that the forward model policy is doing better on the Striker, this is just a consequence of reward hacking; the agent learns a single movement that pushes away the ball without considering the target position. In contrast, the policy learned using NAS aims for the right direction but does not push it hard enough to make it all the way to the target. This happens when, following a random policy in the target domain, we do not record enough interaction between the robot and the ball to model this behavior correctly. An expert policy (e.g. given by human demonstration) could help alleviate this issue and make sure that the relevant parts of the state action space are covered when collecting the data. Altogether it highlights the fact that we cannot hope to learn a good transfer policy for states where there exists both a significant discrepancy between the source and target environments and insufficient data to properly train the LSTM.

We also varied the number of trajectories used to train the model in NAS to see how it influences the final performance of the transferred policy, see Figure 6. When more than 100 trajectories are used, the difference with the expert policy is not visually noticeable and the difference in reward is only due to variance in the evaluation and policy learning. This is in contrast to the 3-4k trajectories required by the expert policy during training to reach its final performance.

4.4 Physical Robot Experiments Overview


(a) Physical Robots

(b) Simulation

Figure 7: The *ErgoShield* environment in reality and simulation. The attacker (sword, left side) has to hit the randomly-moving defender (shield, right side) as often as possible in 10s. In the left image the defender robot is covered in a sock to mitigate the attacker getting stuck too often. The joint angles were compensated for backlash in the left image to highlight similarities.

We use the existing Poppy Ergo Jr robot arm in a novel task called *"ErgoShield"*, a low-cost physical variant of the OpenAI Gym *"Reacher"* task. Each robot arm has 6 DOF: with respect to the robot's heading 1 perpendicular joint at the base, followed by 2 in parallel, 1 perpendicular, and two more in parallel. All joints can rotate from -90 degrees to +90 degrees from their resting position.

For this task we fixed two of these robots onto a wooden plank, facing each other at a distance that left the two tips 5mm apart in resting position. One robot is holding a "sword" (pencil) and the other is holding a shield. The shield robot ("defender") moves every 2.5s to a random position in which the shield is in reach of the opponent. The sword robot ("attacker") is the only robot directly controllable. Each episode lasts 10s and the attacker's goal is to touch the shield as often as possible. Every time a hit is detected, the robots reset to a resting position (attacker) and different random position (defender). The setup is depicted in Figure 7 and additional specifications can be found in appendix B. This environment is accompanied by a simulation in PyBullet[5][6].

The environment can be controlled at 100Hz by sending a 6-element vector in range [-1,1] corresponding to the desired motor position. The environment is observed at 100Hz as a 24-element vector consisting of: attacker joint angles, attacker joint velocities, defender joint angles, defender joint velocities. In the physical robots, the hit detection is implemented by coating both sword and shield in conductive paint, to which a MakeyMakey[7] is attached.

For the offline data collection, a single robot with an empty pen holder end-effector was instructed to move to 3 random positions for one seconds each (corresponding to an episode length of 300 frames). We collected 500 such episodes equivalent to roughly half an hour of robot movement including resetting between episodes.

4.5 Results on Sim-to-Real Transfer

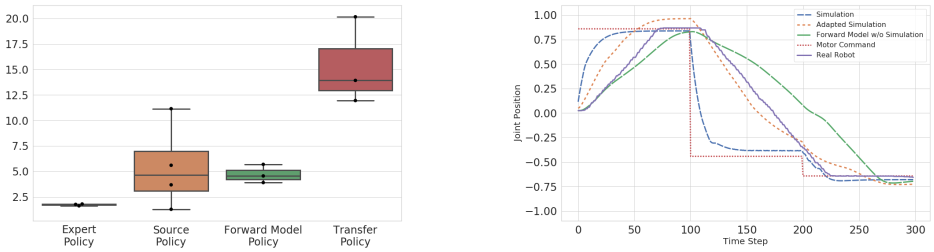
We followed the same experimental paradigm as in the sim-to-sim experiments in 4.1: 3 expert policies were trained directly on the real robot, 3 policies were trained in simulation and were evaluated on the real robot, 3 policies were trained using our method, and 3 policies were trained with only the forward dynamics model. All policies were trained with the same PPO hyperparameters, save for the random seed. The hyperparameters can be found in appendix C.

The evaluation results are displayed in Figure 8a. The policies trained directly on the real robot performed significantly worse than all the other policies. The main reasons for this are (a) the hit

[5] `https://pybullet.org/wordpress/`

[6] The simulated environment is also available on GitHub at `https://github.com/fgolemo/gym-ergojr`

[7] `https://makeymakey.com/`



(a) Method Comparison (Real Robot)

(b) Example Single Joint Position and Estimate over Time

Figure 8: Results of different simulation to real transfer approaches. Left: comparison of average rewards of 20 rollouts of 3 policies per approach. Right: comparison of single joint behavior when receiving a target position (violet dotted) in simulation (green dashed), on the real robot (blue solid), and estimates from the forward model (red, dashed) and our method (yellow, dotted)

detection is not perfect (as in simulation) and since not every hit gets detected the reward is sparser[8]. And (b) since the attacker robot frequently gets their sword stuck in the opponent, in themselves, or in the environment, exploration is not as easy as in simulation.

We did not find a significant difference in the performance of the simulation and forward dynamics policies. However, our approach (the "Transfer policy") yielded a significantly better results than any others.

Figure 8b (and in more detail appendix D) shows for a single joint how the different approaches estimate the joint position under a given action. The simulation approaches the target position asymptotically while the real robot approaches the same value linearly. It is worth noting that even though the forward model can estimate the recorded dataset very well, policies trained using only this model and no simulation tend to perform badly. This is likely due to the forward model overfitting to the training data and not generalizing to other settings. This is a crucial feature of our method: by augmenting the simulator we are able to utilize the same learned augmentation to different tasks.

5 Conclusion

Currently, deep reinforcement learning algorithms are limited in their application to real world scenarios by their high sample complexity and their lack of guarantees when a partially trained policy is deployed. Transfer learning from simulation-trained agents offers a way to solve both these issues and enjoy more flexibility. To that end, we introduced a new method for learning a model that can be used to augment a robotic simulator and demonstrated the performance of this approach as well as its sample efficiency with respect to real robot data.

Provided the same robot is used, the model only has to be learned once and does not require policy-specific fine-tuning, making this method appropriate for multi-task robotic applications.

In the future, we would like to extend this approach to more extensive tasks and different robotic platforms to evaluate its generality, since working purely in simulation leaves some noise that occurs in real robots unmodeled. We would also like to move to image-based observations because our current action/observation spaces are low-dimensional but have a very high frequency (50Hz in sim, 100Hz on real robot). Since our approach is already neural network-based and neural networks are known to scale well to high dimensionality, this addition should be straightforward.

Another interesting path to investigate would be to combine more intelligent exploration methods for collecting the original dataset. If the initial exploration is guided by intrinsic motivation or count-based exploration it might further improve the sample-efficiency and reduce the amount of random movements that need to be recorded in the real robot.

[8] There are no false positives, but we estimate that $10 - 15\%$ hits aren't recognized.

Acknowledgments

This work was made possible with the funding and support of CIFAR, CHIST-ERA IGLU, and ANR. The authors would like to thank the MILA lab for being an amazing research environment and the FLOWERS team at INRIA Bordeaux for the ongoing support with the robots. On top of that, the authors would like to thank David Meger for providing a home for the little robots and input on the project, Herke van Hoof for giving valuable advice on the training procedure, and Alexandre Lacoste for making us revisit the method several times over until we were crystal clear. Additional thanks to NVIDIA for providing a Geforce Titan Xp for the INRIA Bordeaux lab.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[5] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 6, 2017.

[6] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*. IEEE, 2018.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] J. C. Zagal, J. Ruiz-del Solar, and P. Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. *IFAC Proceedings Volumes*, 37(8):834–839, 2004.

[10] J. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 57–62, 2004.

[11] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126. ACM, 2010.

[12] J. C. Zagal and J. Ruiz-Del-Solar. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, 50(1):19–39, 2007.

[13] J. C. G. Higuera, D. Meger, and G. Dudek. Adapting learned robotics behaviours through policy adjustment. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5837–5843. IEEE, 2017.

[14] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995.

[15] A. Punjani and P. Abbeel. Deep learning helicopter dynamics models. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3223–3230. IEEE, 2015.

[16] J. Fu, S. Levine, and P. Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4019–4026. IEEE, 2016.

[17] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 242–248. IEEE, 2016.

[18] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[19] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[20] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[21] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[22] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[23] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[24] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

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

[26] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[27] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017.

[28] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*, 2017.

[29] J. Hanna and P. Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*, 2015.

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

[32] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[33] M. Lapeyre, P. Rouanet, J. Grizou, S. Nguyen, F. Depraetre, A. Le Falher, and P.-Y. Oudeyer. Poppy project: open-source fabrication of 3d printed humanoid robot for science, education and art. In *Digital Intelligence 2014*, page 6, 2014.

A Experiment details

A.1 Neural Network

For the Pusher, the neural network architecture is a fully connected layers with 128 hidden units with ReLU activations followed by a 3 layers LSTM with 128 hidden units and an other fully connected layers for the outputs. The Striker share the same architecture with 256 hidden units instead. The ErgoReacher environment only needed an LSTM of 3 layers with 100 hidden units each. Networks are trained using the Adam optimizer for 250 epochs with a batch size of 1 and a learning rate of 0.01. We use the PPO implementation and the recommended parameters for Mujoco from https://github.com/ikostrikov/pytorch-a2c-ppo-acktr

A.2 Why Backlash?

Regarding the setup for the sim2sim experiments, we tried increasing the reality gap between the two simulated environments on different simulators (Gazebo, MuJoCo, Unity) by tuning physical parameters like mass, inertia and friction. However, we found that small changes in these properties did not affect the policy trained in the source domain, whereas large changes made the environment unstable and it was not possible to learn a good policy directly in the target environment. Nevertheless, even in this extreme case the message from Figure 5 still holds and the proposed method was doing better than the forward model. We then settled on backlash to solve the previous issues as it offered a good compromise between simulation stability and inter-simulation difference.

As an example of one of these preliminary experiments, we increased the mass and inertia of the arm on the Pusher task by 50%, increased the mass and inertia of the pushable object by 100% (i.e. doubled it), and increased the surface friction of the table by 500% while keeping backlash. We found that with these changes, the difference model was still doing much better than the forward model. Looking at a trajectory rollout, the error was significant but close to zero. It should be noted, that such changes created a significant difference between the source and target environments. The new expert policy only averaged -91.3 in reward instead of the previous -67.5 (calculated over 100 episodes on 5 different seeds) which shows the reliability of our method when the reality gap increases.

B ErgoShield Specifications

The "swords" we used are standard BIC 19.5cm wooden pencils. The pencil is sticking out of the penholder end-effector by about 2cm at the bottom. The shield is 2.35cm in radius and 0.59cm deep in the center. A 3D-printable version is available at https://www.tinkercad.com/things/8UXdY4a5xdJ-ergo-jr-shield#/.

The random sampling ranges in degrees on each joint of the shield robot are $[45, 15, 15, 22.5, 15, 15]$ respectively centered around the resting position.

The connection for training the real robot live has an average latency of 10ms. On the real robot the noise and non-stationarities can stem from

- The gear backlash on the Dynamixel XL-320 servos.
- The body of the robot being 3D-printed out of thin PLA plastic.
- The body parts being mounted to the servos via plastic rivets.
- Overheating of the motors and wear of the body parts.
- The electro-conductive paint thinning out over time.

C PPO Hyperparameters Real Robot

Parameter	**Value**
algo	ppo
entropy-coef	0
gamma	0.99
lr	3e-4
num-frames	1000000
num-mini-batch	32
num-processes	4
num-stack	1
num-steps	2048
ppo-epoch	10
seed	[0,1,2]
tau	0.95
use-gae	True

D Quantitative Analysis of Trajectories

Table 1 displays the differences between the expert policy (the policy rolled out on the real robot) and (a) the source policy (same policy in simulation), (b) the forward model policy (the policy rolled out through the LSTM), and (c) the transferred model policy (our approach for modifying the simulator output).

	1st Quartile	**Mean**	**3rd Quartile**
(a) Expert-Source	0.027	0.120	0.172
(b) Expert-Forward Model	0.002	0.008	0.009
(c) Expert-Transferred M.	0.015	0.063	0.078

Table 1: Quantitative analysis of trajectory differences on the ErgoShield task, calculate by the sums of squared differences at each point in the trajectory over 1000 trajectories.

The point-wise difference in (a) indicates the expected significant deviations between simulation and real robot. The low deviations in (b) are specifically what that model was trained for and are therefore near-zero. In practice however, this leads to overfitting, i.e. the so-trained model doesn't perform well on policies for which this model has not been exposed to any trajectories (which is evident from the performance in Figure 8a). The differences in (c) show that the modified simulator is closer to the real robot trajectory. In combination with the final performance in Figure 8a we can infer that our model does not overfit as the forward model does because it's harnessing the simulator to generalize to previously unseen trajectories.