Sim-to-Real in Reinforcement Learning Across Multiple Domains

Tim Knudsen

FH Wedel University of Applied Sciences
tim04knudsen@gmail.com

May 25, 2025

Abstract

Reinforcement Learning (RL) has achieved remarkable success in simulated environments. However, independent of algorithmic performance, deploying RL policies in the real world remains challenging due to discrepancies between simulation and reality. This paper provides a structured overview of key challenges in sim-to-real transfer, categorized by typical mismatches in observations, actions, transitions, and rewards. We systematically review and compare techniques such as domain randomization, system identification, grounded action transformation, and learned reward functions. For each category, we highlight representative approaches, discuss their strengths and limitations, and reference empirical case studies that demonstrate their practical impact.

I. Introduction

Reinforcement Learning (RL) has achieved remarkable results in simulated environments (e.g. video games, simulated robotics) but often fails to generalize to real-world systems due to the *sim-to-real gap*. Simulation can provide abundant and safe data, yet differences in visual perception, physical dynamics, and task specifics cause learned policies to degrade in reality [16, 13]. For instance, a policy that navigates hallways with randomized CAD renderings can transfer to a real drone (CAD2RL [13]), and a humanoid robot hand can solve a Rubik's cube in reality after training with Automatic Domain Randomization (ADR) in simulation [1]. Nevertheless, open challenges remain: how to systematically reconcile each aspect of the Markov Decision Process (observations, actions, transitions, rewards) between sim and real.

This paper revises and streamlines the literature through the Observation–Action–Transition–Reward (OATR) lens. We define each MDP component and describe how discrepancies in that component contribute to the transfer challenge. We then review sim-to-real methods aligned to each OATR category, emphasizing practical examples and empirical findings (e.g. Fig. 1, Fig. 4). In each section, we critically evaluate techniques (such as domain randomization versus system identification for dynamics) and highlight gaps. Our goal is to provide a clear, concise master's-level overview: by focusing on OATR, readers can identify which aspects of sim or real must be bridged and which methods apply. The remainder is organized into Observation-, Action-, Transition-, and Reward-focused transfer.

II. Background

i. Definition

An RL task is an MDP characterized by observation (state) s, action a, transition $T(s, a \to s')$, and reward r. In simulation, these are often idealized: perfect sensors, discretized actions, deterministic physics, and easily computable

rewards. In reality, by contrast, sensors produce noisy observations, actuators have latency and limits, dynamics are stochastic and hard to model, and rewards may be sparse or indirect. Table 1 summarizes typical sim-versus-real discrepancies. Any of these mismatches can cause failure: for example, a vision policy may overfit simulated textures and fail on real images, or an action sequence learned without actuator latency may be unsafe on hardware.

Table 1: *Simulated vs real MDP components: sources of discrepancy (simulated vs real).*

Observation:	Idealized vs noisy vision/depth; synthetic vs real textures (rendering)
Action:	Precise, discrete commands vs actuator noise, latency, continuous control
Transition:	Deterministic physics vs stochastic dynamics, friction, contact variations
Reward:	Engineered or dense reward vs sparse/uncertain task objectives

Each sim-to-real method typically addresses one or more of these elements. For instance, *domain randomization* perturbs visuals and/or physics during training to cover a wide range of possible observations or transitions [16, 13]. In contrast, *system identification* methods use real data to calibrate the simulator's parameters [3]. Some approaches combine both: e.g. OpenAI's ADR gradually expands randomization range, effectively automating calibration while training [1]. We now detail techniques by OATR component, starting with observations.

III. Observation-Level Transfer

Observation-level sim-to-real focuses on aligning perception between sim and reality. The dominant strategy is *visual domain randomization*, which artificially varies rendering during training (textures, lighting, colors, camera pose) so that the policy learns invariant features [13, 16]. For example, Sadeghi and Levine's CAD2RL rendered randomized indoor scenes for a quadrotor, enabling a monocular vision policy to generalize to real hallways (without any real images for training)[13]. Figure 1(A) illustrates how ADR similarly randomizes environments (OpenAI Rubik's Cube example): the learned vision encoder and policy must cope with varied camera views and object appearances. Domain randomization is popular because it requires no real data and often suffices if the randomized range covers the real-world variability. However, it can lead to inefficiency or failure if unrealistic combinations are included or if the real domain lies outside the chosen randomization range.

Beyond randomization, *domain adaptation* methods use learning to map between sim and real observations. For example, generative techniques (e.g. CycleGAN or SimGAN) can transform real camera images into the simulator's style, or vice versa, and then apply a sim-trained policy[17]. These methods may require some real data or paired images, which partially defeats the pure sim-training advantage. In practice, adaptation is more common in perception (e.g. object detection) than in end-to-end RL due to the difficulty of adversarial training in policy learning.

Another approach is *sensor modeling and augmentation*: simulating sensor noise, delays, or partial observability in training. For example, adding realistic noise to depth or lidar readings, or dropping out camera pixels, can make a learned policy robust to sensor imperfections. Similarly, using multi-modal inputs (e.g. combining vision and proprioception) can reduce reliance on any single sim-only signal. Some works design a compact observation space to eliminate noisy dimensions: Tan et al. found that excluding the robot's yaw angle (which drifted) improved transfer for quadruped control[15].

Overall, observation-level transfer is well-served by domain randomization and careful sensor simulation. These methods, combined with powerful neural encoders, allow policies to generalize across texture and light-

ing variations[16, 13]. Figure 1(A) illustrates this concept: the vision system is trained on many randomized versions of the cube and hand. We will use Fig. 1 again below to show how those observations feed into the policy and state estimator.

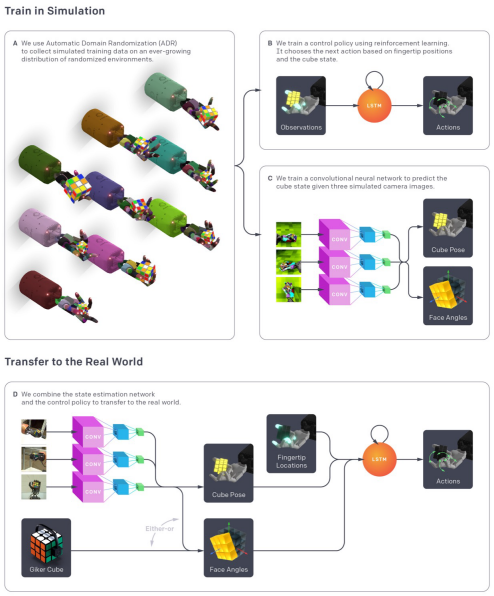


Figure 1: *Training pipeline with Automatic Domain Randomization (ADR) for sim-to-real transfer. (A) Simulation generates randomized cube-and-hand images; (B) a policy (LSTM) maps simulated observations (hand and cube states) to actions; (C) a CNN state estimator is trained to predict cube pose from images; (D) during real-world deployment, camera images pass through the CNN, then policy acts on estimated state.* (Adapted from *Akkaya et al. [1].)*

Evaluation. Domain randomization for visual perception has shown robust performance in tasks such as drone navigation (CAD2RL [13]) and robotic manipulation (Dactyl [1]). Its main advantage is that it requires no real-world data and can be scaled easily. However, its effectiveness is sensitive to the diversity and realism of the randomized features. Over-randomization or unrealistic textures can degrade learning efficiency. Domain adaptation methods offer finer alignment but typically require real data and are challenging to scale to end-to-end RL pipelines [17].

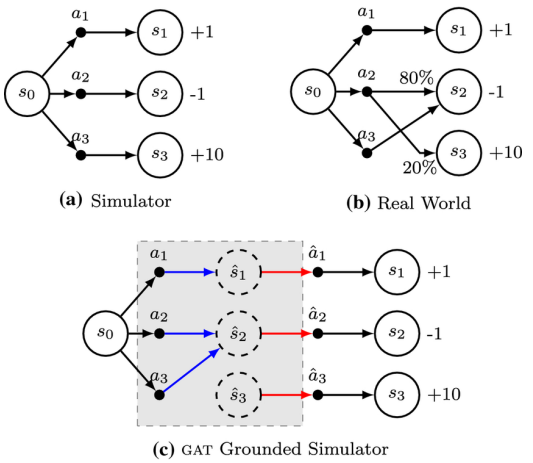
IV. Action-Level Transfer

Action-level differences stem from how commands translate to real actuators. Simulators often assume ideal, synchronous action execution, whereas real robots have continuous-time dynamics, delays, saturations, and noise. Key challenges include actuator latency, unmodeled torque limits, and quantization (e.g. servos that only accept discrete commands).

A straightforward remedy is to *model the actuator dynamics and noise* in simulation. For instance, injecting random motor noise, friction variation, and delays during training produces policies that are robust to these effects. OpenAI's Dactyl system (dexterous hand manipulation) used extensive randomization of physical properties (finger friction, motor strength, mass of objects) so that the learned policy could tolerate real actuator uncertainties[1]. In legged locomotion, Tan et al. randomized actuator response times and motor parameters, which improved real-world stability[15]. This form of domain randomization on the action channel hedges against mismatch by exposing the policy to a range of possible actuator behaviors.

Beyond randomization, more principled methods attempt to *transform actions* between domains. *Grounded Action Transformation* (GAT) learns a mapping that "grounds" simulator actions to mimic real-world effects[8]. Essentially, one learns a function $g(a)$ such that executing a in sim has the same outcome as $g(a)$ in real, or vice versa. Variants like SGAT (stochastic GAT) and RGAT (reinforced GAT) extend this idea to handle stochasticity or use RL to learn the transformation[5, 9]. These methods explicitly adjust the transition function of the simulator to better match the real robot's response. For example, if a real robot joint moves slower than the simulator model, GAT would scale down the sim commands so that actions are comparable. **Figure 2** illustrates this principle: a forward model predicts the real-world outcome, which is then inverted through the sim-

ulation model to generate a grounded action. In practice, grounded actions require some real data for calibration, but they can significantly reduce the sim-real control mismatch.



(a) Simulator **(b)** Real World

(c) GAT Grounded Simulator

Figure 2: *Illustration of the Grounded Action Transformation (GAT) mechanism: A forward model f_{real} predicts the next real-world state, which is then mapped back into the simulator action space via the inverse simulation model f_{sim}^{-1}. This transformation aims to produce simulator actions that better match real-world effects.* (Source: *Hanna et al., 2021)*

Another complementary technique is *robust policy design*, which trains agents to tolerate uncertainty. This includes adversarial training with perturbed actions or robust control frameworks with formal guarantees. Policies trained with adversarial noise tend to ignore small deviations in command execution, improving resilience to mismatches.

In summary, action-level transfer blends randomized modeling and explicit calibration. Randomization (as in Fig. 1(B)) can cover many uncertainties but may not capture systematic biases (e.g. consistent actuator lag). Grounded action techniques and system identification fill this gap by learning the actual mapping, at the cost of requiring real observations. We encourage combining both: randomize broadly, then fine-tune or adapt the action model with real data for best performance.

Evaluation. Injecting actuator noise and delays during training improves robustness, as shown in legged locomotion by Tan et al. [15]. However, this approach may not fully capture systematic real-world effects such as persistent latency or actuator limits. Grounded Action Transformation (GAT) and its variants [8, 5] offer precise domain alignment, but depend on real-world calibration and are often task-specific. In practice, combining randomized actuation with post-hoc transformation yields better sim-to-real transfer in continuous control settings.

V. Transition-Level Transfer

Transition-level sim-to-real deals with dynamics modeling: how the state evolves under actions. The simulator's physics (mass, friction, contacts) rarely match reality precisely, especially in contact-rich or deformable tasks. This is often the hardest gap to bridge, since even small physics errors can accumulate.

A common strategy is *dynamics randomization*: sampling simulation parameters (mass, inertia, friction coefficients, gravity, etc.) from a broad distribution during training. This approach generates a policy robust to a variety of dynamics. It was famously used by OpenAI in their bipedal robot and Dactyl hand experiments, randomizing parameters like joint friction, motor strength, and even gravity to ensure the policy did not rely on any precise setting[1]. Tan et al. similarly randomized terrain friction and actuator gains for quadruped locomotion, leading to successful transfer of galloping and trotting gaits[15]. Figure 3 shows an example: a Minitaur robot learned to gallop in simulation and executed the gait in reality. While not a plot, this juxtaposition of sim vs real performance (from Tan et al.) illustrates that well-randomized training can produce controllers that behave similarly in both domains.

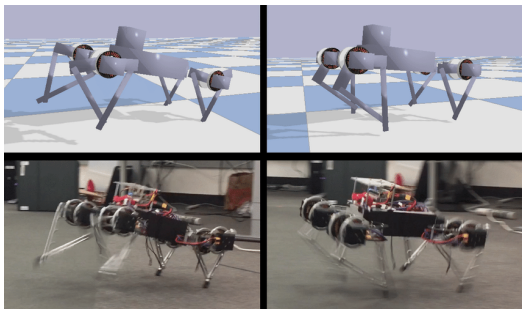


Figure 3: *Quadruped locomotion learned in simulation and deployed on hardware. (Top) Simulated Minitaur robot in galloping gait (four frames at different times). (Bottom) The real Minitaur performing a learned gallop. (*Source: *Tan et al. [15].)*

However, randomization has limits: if the real-world dynamics lie outside the randomized range, or if the gap is *irreducible* (e.g. complex contact physics that simulators cannot represent), performance suffers. Moreover, randomization offers little guidance for refining sim parameters post-transfer. An alternative is *system identification (SysID)*: using real-world data to calibrate the simulator. Early SysID efforts involved manual tuning of physical parameters. Recent approaches, such as SimOpt [3], automate this loop: a policy is trained in sim, tested on hardware, and the parameter distribution is updated to reduce trajectory mismatch. Grounded Simulation Learning [8] further refines transitions by inserting learned corrections directly into the simulation dynamics.

More recently, *task-driven adaptation* frameworks aim to optimize simulation not for physical realism but for real-world task performance. AdaptSim [12] exemplifies this approach. It alternates between training a policy in simulation, testing on hardware, and adjusting the simulation distribution via a learned adaptation policy. The result is a task-driven refinement of simulation. Figure 4 (adapted from Ren et al.) illustrates this in a real scooping task, where each iteration significantly improves success rate.

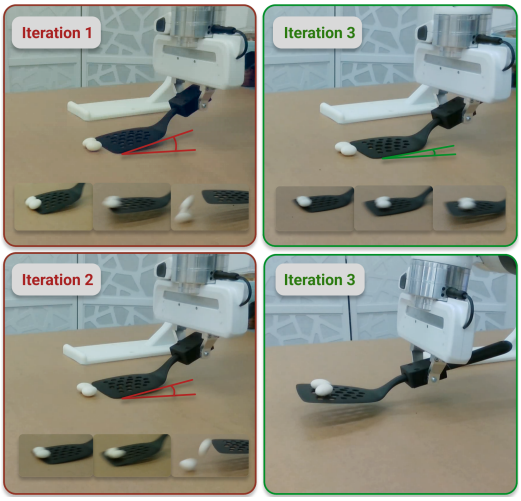


Figure 4: *Adaptive simulation improvement on a real scooping task (Ren et al. [12]). The robot attempts to scoop marshmallows with a spatula under varying simulation parameters. Iteration 1 (red) shows poor performance; by Iteration 3 (green) the learned policy successfully scoops the object consistently. (Credit: Adapted from Ren et al. [12].)*

In summary, transition-level transfer methods range from broad dynamics randomization to data-driven calibration. Task-driven approaches like AdaptSim aim to optimize simulation directly for real-world task performance. Across studies, hybrid pipelines—starting with randomization and refining via adaptation—consistently outperform isolated strategies [3, 12].

Evaluation. Dynamics randomization has proven effective in locomotion and manipulation tasks (e.g., OpenAI Dactyl [1], Minitaur robot [15]). However, it assumes that the true real-world parameters are covered by the randomized distribution, which is not always guaranteed. System identification methods such as SimOpt [3] provide more accurate sim-real alignment by directly minimizing discrepancy through real-world trials. While this improves transfer performance, it requires repeated access to hardware and assumes stable and measurable dynamics.

Table 2: *Comparison of Transition-Level Transfer Methods*

Method	Real-World Data	Adaptability	Simulation Fidelity	Examples
Domain Randomization	✗	Broad	Moderate	Dactyl [1], Tan et al. [15]
System Identification	✓	Precise	High	SimOpt [3]
AdaptSim	✓ / *	Task-Driven	High	Ren et al. [12]

VI. Reward-Level Transfer

The reward function reflects the task objective and is often overlooked in sim-to-real discussions. In simulation, one can design dense rewards (e.g. penalize distance to goal at each step). In reality, measuring the true task success may be harder (e.g. it may require expensive sensors or human input). Furthermore, simulators sometimes use artificial knowledge (like exact pose or contact flags) to compute reward that are unavailable in real deployment.

A simple tactic is to avoid reward mismatch by using task goals that can be measured in both domains. For example, if the task is goal-reaching, use the distance of an end-effector (which can be tracked by real sensors) as the reward. If a reward requires internal state (like velocity in sim), one can approximate it using onboard sensors or drop it from the function. Another strategy is to degrade the reward signal during training (add noise or delays) to simulate real sensing imperfections.

Recent research increasingly leverages *learned reward functions* to address the limitations of hand-crafted design. Instead of manually specifying the objective, reward models are trained from data, either through supervised learning or auxiliary objectives. Methods such as self-supervised reward learning [6] or consistency-based techniques [14] allow agents to generate internal learning signals without ground-truth rewards. These approaches have been applied successfully in tasks such as visual navigation and robotic manipulation.

Inverse Reinforcement Learning (IRL) provides a principled framework for inferring reward functions from expert behavior [11]. Instead of defining a reward function directly, the agent is shown successful demonstrations and attempts to recover a reward that explains them. Recent methods such as Adversarial IRL (AIRL) [7] model this as a game between policy and discriminator, improving scalability and generalization. Applications include robotic manipulation and autonomous driving. However, IRL methods often require large, high-quality demonstration datasets and are sensitive to noise in the trajectories.

Preference-Based Reinforcement Learning (PbRL) offers a more data-efficient alternative by using binary comparisons instead of full expert demonstrations. Instead of labeling trajectories with reward values, a human selects which of two behaviors is preferred. These comparisons are used to train a reward model, which is then optimized. Notable examples include D-REX [2] and PEBBLE [10], which have shown strong results in sparse-reward environments like Atari and robotics. OpenAI's preference-based approach enabled a simulated agent to learn a backflip using fewer than 1,000 bits of human feedback [4]. Similarly, Deep TAMER [18] showed that just minutes of real-time interaction can outperform traditional RL in games like Bowling. Despite their efficiency, these methods are sensitive to feedback consistency and suffer from potential bias.

Compared to discrepancies in observations, actions, and transitions, reward-level gaps are more task-specific and harder to generalize. While fewer sim-to-real pipelines explicitly address reward transfer, its importance is growing as RL moves toward real-world deployment. Hybrid approaches that combine learned reward models, preference feedback, and limited real-world supervision are a promising direction.

Evaluation. Reward design remains one of the most challenging aspects of sim-to-real

Table 3: *Comparison of Reward-Level Transfer Methods*

Method	Real-World Feedback	Sample Efficiency	Generalization	Examples
Inverse RL (IRL)	✓	Low	Moderate	AIRL [7]
Preference-Based RL (PbRL)	*	High	High	Christiano et al. [4]
Self-Supervised Rewards	✗	High	Task-Specific	Ze et al. [19]

transfer. Self-supervised and learned rewards help avoid reliance on inaccessible ground-truth signals but often lack interpretability. Inverse reinforcement learning (IRL) approaches, such as AIRL [7], perform well when expert demonstrations are available but are difficult to scale and sensitive to trajectory quality.

VII. Discussion and Outlook

Using the OATR framework clarifies that sim-to-real transfer is multi-faceted. Domain randomization has emerged as a powerful unifying idea: it can be applied to observations (textures/colors), actions (motor noise), and transitions (dynamics parameters). Its strength is simplicity and applicability without real data[16, 13, 1]. However, it lacks guarantees and may require careful tuning of parameter ranges. System identification and grounded action approaches, by contrast, leverage real-world feedback to adapt the simulator or policy, which often leads to more precise sim-real alignment[3, 8, 12]. These tend to perform better on specific tasks but depend on collecting sufficient real experience.

Figure 1 through 4 summarize illustrative results: learning in simulation with domain randomization enabled OpenAI's robot hand to handle varied physical conditions, as shown in the ADR pipeline (Fig. 1); Tan et al.'s randomized locomotion policy succeeded on a real quadruped (Fig. 3); and AdaptSim's iterative adaptation dramatically improved real-world scooping (Fig. 4). These examples underscore that combined strategies often work best: randomize broadly, then refine with targeted real data.

In practice, successful sim-to-real transfer often involves *engineering* trade-offs. For example, one may choose which aspects of the simulator to increase fidelity (e.g. detailed actuator model) and which to randomize.There is evidence that focusing on the most significant discrepancies yields the most benefit. Future work may further integrate these techniques; e.g. meta-learning frameworks that automatically determine which parameters to randomize versus calibrate.

Meta-Evaluation and Strategy. While each method addresses a distinct element of the sim-to-real pipeline, their practical value often hinges on real-world feasibility, data availability, and task constraints. Observation-level techniques such as domain randomization offer strong generalization, especially for vision-based agents, but may suffer from limited realism or coverage gaps. Transition- and action-level transfer methods like system identification or grounded transformations yield higher precision but are inherently hardware-dependent.

Reward-level approaches represent the most fragile layer: while learned rewards, inverse RL, and preference-based methods reduce engineering burden, they are often unstable and require high-quality supervision signals. Across the reviewed literature, hybrid strategies emerge as the most effective: initial training with broad randomization followed by targeted adaptation (e.g., AdaptSim [12]) leads to the most consistent performance gains.

To systematically compare these techniques, future work should establish benchmark scenarios with well-defined evaluation metrics across domains. There is also a need for real-time adaptive pipelines that dynamically adjust simulation parameters and reward structures based on observed sim-real discrepancies—an open challenge for sim-to-real research at scale.

VIII. Conclusion

Sim-to-real transfer in RL requires carefully addressing mismatches in observations, actions, transitions, and rewards. Organized by the OATR framework, we have reviewed the key methods and their trade-offs. Domain randomization can yield robust policies by training on diverse simulated variations, while system identification and adaptive methods use real data to fine-tune the simulation. Our discussion and selected examples from the literature highlight both the progress made (e.g. OpenAI's ADR, Tan et al.'s locomotion) and the remaining challenges (e.g. reward gaps, irreducible dynamics differences). By critically evaluating each approach, we hope to provide a clear resource for researchers to design sim-to-real pipelines. Ongoing research will continue to blend these techniques and exploit new ideas (like learned simulators or real-time adaptation) to further bridge the reality gap.

References

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, and W. Zaremba. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning (ICML)*, 2019.

[3] Y. Chebotar, A. Handa, K. Hausman, S. Schaal, G. S. Sukhatme, and O. Kroemer. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[4] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[5] M. Desai, A. Walsh, A. Quillen, and K. Choi. Stochastic grounded action transformation for sim-to-real transfer, 2020. arXiv preprint arXiv:2002.11957.

[6] B. Eysenbach, R. Salakhutdinov, and S. Levine. Replacing rewards with examples: Example-based policy search via recursive classification. *International Conference on Learning Representations (ICLR)*, 2021.

[7] J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[8] J. Hanna and P. Stone. Grounded action transformation for robot learning in simulation. *Artificial Intelligence*, 247:45–65, 2017.

[9] S. Karnan, R. Kumarasubramanian, and C. Xu. Reinforced grounded action transformation for sim-to-real transfer, 2020. arXiv preprint arXiv:2012.02353.

[10] K. Lee, S. Lee, K. Lee, and H. Lee. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. In *International Conference on Machine Learning (ICML)*, 2021.

[11] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

[12] A. Ren, H. Dai, B. Burchfiel, and A. Majumdar. Adaptsim: Task-driven simulation adaptation for sim-to-real transfer. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2023.

[13] F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[14] A. Sharma, D. Kalashnikov, and S. Levine. State-only imitation learning from observations via latent space classification. *Conference on Robot Learning (CoRL)*, 2022.

[15] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[16] 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. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[17] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. *arXiv preprint arXiv:1709.10163*, 2017.

[19] Y. Ze, N. Hansen, Y. Chen, M. Jain, and X. Wang. Visual reinforcement learning with self-supervised 3d representations. *arXiv preprint arXiv:2210.07241*, 2023.