Machine Learning (2021) 110:2469–2499
https://doi.org/10.1007/s10994-021-05982-z

Grounded action transformation for sim-to-real reinforcement learning

Josiah P. Hanna[1] · **Siddharth Desai**[2] · **Haresh Karnan**[2] · **Garrett Warnell**[3] · **Peter Stone**[4]

Received: 9 March 2020 / Revised: 30 September 2020 / Accepted: 12 April 2021 /
Published online: 13 May 2021

Abstract
Reinforcement learning in simulation is a promising alternative to the prohibitive sample cost of reinforcement learning in the physical world. Unfortunately, policies learned in simulation often perform worse than hand-coded policies when applied on the target, physical system. Grounded simulation learning (GSL) is a general framework that promises to address this issue by altering the simulator to better match the real world (Farchy et al. 2013 in Proceedings of the 12th international conference on autonomous agents and multiagent systems (AAMAS)). This article introduces a new algorithm for GSL—Grounded Action Transformation (GAT)—and applies it to learning control policies for a humanoid robot. We evaluate our algorithm in controlled experiments where we show it to allow policies learned in simulation to transfer to the real world. We then apply our algorithm to learning a fast bipedal walk on a humanoid robot and demonstrate a 43.27% improvement in forward walk velocity compared to a state-of-the art hand-coded walk. This striking empirical success notwithstanding, further empirical analysis shows that GAT may struggle when the real world has stochastic state transitions. To address this limitation we generalize GAT to the *stochastic* GAT (SGAT) algorithm and empirically show that SGAT leads to successful real world transfer in situations where GAT may fail to find a good policy. Our results contribute to a deeper understanding of grounded simulation learning and demonstrate its effectiveness for applying reinforcement learning to learn robot control policies entirely in simulation.

Keywords Reinforcement learning · Robotics · Sim-to-real · Bipedal locomotion

Editors: Yuxi Li, Alborz Geramifard, Lihong Li, Csaba Szepesvari, Tao Wang.

This work contains material that was previously presented at the 31st AAAI Conference on Artificial Intelligence (AAAI 2017) and the International Conference on Intelligent Robots and Systems (IROS 2020). This article unifies these previous works to comprise a "complete" article. In addition to the previously published work, we have 1) reformulated the presentation of the algorithm, 2) expanded the empirical analysis of the GAT algorithm to include two new tasks on the simulated and physical NAO robot, and 3) conducted a qualitative analysis of the simulator corrections in the two new tasks.

✉ Josiah P. Hanna
josiah.hanna@ed.ac.uk

Extended author information available on the last page of the article

1 Introduction

Manually designing control policies for every possible situation a robot could encounter is impractical. Reinforcement learning (RL) provides a promising alternative to hand-coding skills. Recent applications of RL to high dimensional control tasks have seen impressive successes within simulation (Schulman et al., 2015b; Lillicrap et al., 2015). Unfortunately, a large gap exists between what is possible in simulation and the reality of learning on a physical system. State-of-the-art learning methods require thousands of episodes of experience which is impractical for a physical robot. Aside from the time it would take, collecting the required training data may lead to substantial wear on the robot. Furthermore, as the robot explores different policies it may execute unsafe actions which could damage the robot.

An alternative to learning directly on the robot is learning in simulation (Cutler & How, 2015; Koos et al., 2010). Simulation is a valuable tool for robotics research as execution of a robotic skill in simulation is comparatively easier than real world execution. Robots in simulation can be run unsupervised without fear of them breaking or wearing down. Simulation can often be ran faster than real time or parallelized to increase the speed at which data for RL can be collected. However, the value of simulation learning is limited by the inherent inaccuracy of simulators in modeling the dynamics of the physical world (Kober et al., 2013). As a result, learning that takes place in a simulator is unlikely to improve real world performance.

Grounded Simulation Learning (GSL) is a framework for learning with a simulator in which the simulator is modified with data from the physical robot, learning takes place in simulation, the new policy is evaluated on the robot, and data from the new policy is used to further modify the simulator (Farchy et al., 2013). The work introducing GSL demonstrates the effectiveness of the method in a single, limited experiment, by increasing the forward walking velocity of a slow, stable bipedal walk by 26.7%. This article introduces a new algorithm—Grounded Action Transformation (GAT)—for simulator grounding within the GSL framework. GAT grounds the simulator by modifying the robot's actions as they are passed to the simulator to, in effect, create a simulator with different dynamics. The grounding function is learned with a small amount of real world and simulated data, allowing the simulator to be modified with less reliance on manual system identification. Additionally, by modifying the simulated robot's actions we can treat the simulator as a black-box and do not require access to change internal parameters of the simulator.

As a first step, in order to facilitate extensive evaluations, we fully implement and evaluate GAT on two tasks using a high-fidelity simulator as a surrogate for the real world. The results of this controlled study contribute to a deeper understanding of transfer from simulation methods and the effectiveness of GAT. We then present two examples of using GAT for sim-to-real transfer of bipedal locomotion policies learned in simulation to a real humanoid robot. In contrast to prior work (Farchy et al., 2013), one task in our real-world evaluation starts from a state-of-the-art walking controller as the initial policy, and nonetheless is able to improve the walk velocity by over 43%, leading to what may be the fastest known stable walk on the SoftBank NAO robot.

Furthermore, to better understand situations where GAT may be successful we consider real world environments that have a high degree of stochasticity. We show in simulated environments that GAT may fail to find high performing policies when environment state transitions are noisy. To address this limitation we generalize GAT to the *stochastic* GAT (SGAT) algorithm and show in simulated, stochastic environments that SGAT finds higher

performing policies than GAT. We implement SGAT on the NAO robot and show that we can learn a fast and stable walking policy over a rough surface while GAT fails to find a stable policy.

2 Preliminaries

In this section we formalize the reinforcement learning setting and the problem of sim-to-real learning.

2.1 Notation

We assume the environment is an episodic *Markov decision process* with state set $\mathcal{S}$, action set $\mathcal{A}$, transition function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor γ, and initial state distribution d_0 (Puterman, 2014). We assume that $\mathcal{S} = \mathbb{R}^k$ and $\mathcal{A} = \mathbb{R}^m$ for some $k, m \in \mathbb{N}_+$. We assume that the transition function, P, is unknown and the reward function, r, is known. We use $P(s'|s, a) := P(s, a, s')$ to denote the conditional probability of state s' given state s and action a. P is also sometimes called the environment's *dynamics*. A policy, $\pi : \mathcal{S} \to \mathcal{A}$, is a function mapping states to actions.

The agent interacts with the environment MDP as follows: The agent begins in initial state $S_0 \sim d_0$. At discrete time-step t the agents takes action $A_t = \pi(S_t)$. The environment responds with $R_t := r(S_t, A_t)$ and $S_{t+1} \sim P(\cdot|S_t, A_t)$ according to the reward function and transition function. After interacting with the environment for at most l steps the agent returns to a new initial state and the process repeats. For notational convenience, we will write that all interactions last l steps, though in fact they may end earlier. In the MDP definition, we also include a terminal state, s_∞, that allows the possibility of episodes ending before time-step l. If at any time-step, t, $S_t = s_\infty$, then for all $t' \geq t$, $S_{t'} = s_\infty$ and $R_{t'} = 0$.

Let $h := (s_0, a_0, r_0, s_1, \ldots, s_{l-1}, a_{l-1}, r_{l-1})$ be a *trajectory*. Any policy, π, and MDP, $\mathcal{M}$, induce a distribution over trajectories, $\Pr(H = h|\pi, \mathcal{M})$, where H is a random variable representing a trajectory. Let $R(h) := \sum_{t=0}^{l-1} \gamma^t r_t$ be the *discounted return* of h. We define the *value* of a policy, $v(\pi, \mathcal{M}) := \mathbf{E}[R(H)|H \sim (\pi, \mathcal{M})]$, as the expected discounted return when sampling a trajectory with policy π in MDP $\mathcal{M}$. We are interested in learning a policy, π, for an MDP, $\mathcal{M}$, such that $v(\pi, \mathcal{M})$ is maximized. We wish to minimize the number of actions that must be taken in $\mathcal{M}$ before a good policy is learned, i.e., we desire low sample complexity for learning.

2.2 Learning in simulation

In this article we study reinforcement learning in a simulated environment with the objective that learned policies will perform well in the real world. We formalize this setting as learning a policy, π, in one MDP, $\mathcal{M}_{\texttt{sim}}$, with the objective of maximizing $v(\pi, \mathcal{M})$. The MDP $\mathcal{M}_{\texttt{sim}}$ is the simulator and $\mathcal{M}$ is the real world. Formally, $\mathcal{M}$ and $\mathcal{M}_{\texttt{sim}}$ are identical MDPs except for the transition function P.[1] We use P to denote the transition

[1] A closely related body of work considers how learning can take place in simulation when the observations the agent receives are different from the real world (e.g., rendered images vs. natural images). We discuss this work in our related work section but consider this problem orthogonal to the problem of differing dynamics.

function of the real world and $P_{\texttt{sim}}$ to denote the transition function of the simulator. We make the assumption that the reward function, r, is user-defined and thus is identical for $\mathcal{M}$ and $\mathcal{M}_{\texttt{sim}}$. However, the different dynamics distribution means that for any policy, π, $v(\pi, \mathcal{M}) \neq v(\pi, \mathcal{M}_{\texttt{sim}})$ since π induces a different trajectory distribution in $\mathcal{M}$ than in $\mathcal{M}_{\texttt{sim}}$. Thus, for any π' with $v(\pi', \mathcal{M}_{\texttt{sim}}) > v(\pi, \mathcal{M}_{\texttt{sim}})$, it does *not* follow that $v(\pi', \mathcal{M}) > v(\pi, \mathcal{M})$—in fact $v(\pi', \mathcal{M})$ could be much worse than $v(\pi, \mathcal{M})$. In practice and in the literature, learning in simulation often fails to improve expected performance (Farchy et al., 2013; Christiano et al., 2016; Rusu et al., 2016b; Tobin et al., 2017).

3 Related work

The challenge of transferring learned policies from simulation to reality has received much research attention of late. This section surveys this recent work as well as older research in simulation-transfer methods. We note that our work also relates to model-based reinforcement learning (Sutton & Barto, 1998). However, much of model-based reinforcement learning focuses on learning a simulator for the task MDP (often from scratch) while we focus on settings where an inaccurate simulator is available a priori.

We divide the sim-to-real literature into four categories: simulator modification, simulator randomization or simulator ensembles, simulators as prior knowledge, and sim-to-real perception learning.

3.1 Simulator modification

We classify sim-to-real works that attempt to use real world experience to change the simulator as simulator modification approaches. This category of work is the category most similar to this work.

Abbeel et al. (2006) use real-world experience to modify an inaccurate model of a deterministic MDP. The real-world experience is used to modify $P_{\texttt{sim}}$ so that the policy gradient in simulation is the same as the policy gradient in the real world. Cutler et al. (2014) use lower fidelity simulators to narrow the action search space for faster learning in higher fidelity simulators or the real world. This work also uses experience in higher fidelity simulators to make lower fidelity simulators more realistic. Both these methods assume *random access modification*—the ability to arbitrarily and locally modify the simulated dynamics of any state-action pair. This assumption is restrictive in that it may be false for many simulators especially for real-valued states and actions.

Other work has used real world data to modify simulator parameters (e.g., coefficients of friction) (Zhu et al., 2018) or combined simulation with Gaussian processes to model where real world data has not been observed (Lee et al., 2017). Such approaches may extrapolate well to new parts of the state-space, however, they may fail if no setting of the physics parameters can capture the complexity of the real world. Golemo et al. (2018) train recurrent neural networks to predict differences between simulation and reality. Then, following actions in simulation, the resulting next state is corrected to be closer to what it would be in the real world. This approach requires the ability to directly set the state of the simulator which is a requirement we avoid in this work.

Manual parameter tuning is another form of simulator modification that can be done prior to applying reinforcement learning. Lowrey et al. (2018) manually identify simulation parameters before applying policy gradient reinforcement learning to learn to push an

object to target positions. Tan et al. (2018) perform similar system identification (including disassembling the robot and making measurements of each part) and adding action latency modeling before using deep reinforcement learning to learn quadrapedal walking. In contrast to these approaches, the algorithms we introduce take a data-driven approach to modifying the simulator without the need for expert system identification.

Finally, while most approaches to simulator modification involve correcting the simulator dynamics, other approaches attempt to directly correct $v(\pi, \mathcal{M}_{\texttt{sim}})$. Assuming $v(\pi, \mathcal{M}) = v(\pi, \mathcal{M}_{\texttt{sim}}) + \epsilon(\pi)$, Iocchi et al. (2007) attempt to learn $\epsilon(\pi)$ for any π. Then policy search can be done directly on $v(\pi, \mathcal{M}_{\texttt{sim}}) + \epsilon(\pi)$ without needing to evaluate $v(\pi, \mathcal{M})$. Rodriguez et al. (2019) introduce a similar approach except they take into account uncertainty in extrapolating the estimate of $\epsilon(\pi)$ and use Bayesian optimization for policy learning. Like this work, both of these works apply their techniques to bipedal locomotion. Koos et al. (2010) use multi-objective optimization to find policies that trade off between optimizing $v(\pi, \mathcal{M}_{\texttt{sim}})$ and a measure of how likely π is to transfer to the real world.

3.2 Robustness through simulator variance

Another class of sim-to-real approaches is methods that attempt to cross the reality gap by learning robust policies that can work in different variants of the simulated environment. The key idea is that if a learned policy can work in different simulations then it is more likely to be able to perform well in the real world. The simplest instantiation of this idea is to inject noise into the robot's actions or sensors (Jakobi et al., 1995; Miglino et al., 1996) or to randomize the simulator parameters (Peng et al., 2017; Molchanov et al., 2019; OpenAI et al., 2018). Unlike data driven approaches, such *domain randomization* approaches learn policies that are robust enough to cross the reality gap but may give up some ability to exploit the target real world environment. This problem may be more acute when learning with simple policy representations, as simpler policies may lack the capacity to perform well under a wide range of environment conditions (Mozifian et al., 2019).

A number of works have attempted to combine domain randomization and real world data to adapt the simulator. Chebotar et al. (2019) randomize simulation parameters and use real world data to update the distribution over simulation parameters while simulatenously learning robotic manipulation tasks. Ramos et al. (2019) take a similar approach. Muratore et al. (2018) attempt to use real world data to predict transferrability of policies learned in a randomized simulation. Mozifian et al. (2019) attempt to maintain a wide distribution over simulator parameters while ensuring the distribution is narrow enough to allow reinforcement learning to exploit instances that are most similar to the real world.

Domain randomization produces policies that are robust enough to transfer to the real world. An alternative approach that does not involve randomness is to learn policies that perform well under an ensemble of different simulators (Boeing & Bräunl, 2012; Rajeswaran et al., 2017; Lowrey et al., 2018). Pinto et al., (2017b) simultaneously learn an adversary that can perturb the learning agent's actions while it learns in simulation. The learner must learn a policy that is robust to disturbances and then will perform better when transferred to the real world.

3.3 Simulator as prior knowledge

Another approach to sim-to-real learning is to use experience in simulation to reduce learning time on the physical robot. Cully et al. (2015) use a simulator to estimate fitness values

for low-dimensional robot behaviors which gives the robot prior knowledge of how to adapt its behavior if it becomes damaged during real world operation. Cutler and How (2015) use experience in simulation to estimate a prior for a Gaussian process model to be used with the PILCO (Deisenroth & Rasmussen, 2011) learning algorithm. Rusu et al. (2016a, b) introduce progressive neural network policies which are initially trained in simulation before a final period of learning in the true environment. Christiano et al. (2016) turn simulation policies into real world policies by transforming policy actions so that they produce the same effect that they did in simulation. Marco et al. (2017) use simulation to reduce the number of policy evaluations needed for Bayesian optimization of task performance. In principle, our work could be used with any of these approaches to correct the simulator dynamics which would lead to a more accurate prior.

3.4 Reality gap in the observation space

Finally, while we focus on the reality gap due to differences in simulated and real world dynamics, much recent work has focused on transfer from simulation to reality when the policy maps images to actions. In this setting, even if P and $P_{\texttt{sim}}$ are identical, policies may fail when transferred to the real world due to the differences between real and rendered images. Domain randomization is a popular technique for handling this problem. Unlike the dynamics randomization techniques discussed above, in this setting domain randomization means randomizing features of the simulator's rendered images (Sadeghi & Levine, 2017; Tobin et al., 2017, 2018; Pinto et al., 2017a). This approach is useful in that it forces deep reinforcement learning algorithms to learn representations that focus on higher level properties of a task and not low-level details of image appearance. Computer vision domain adaptation methods can also be used to overcome the problem of differing observation spaces (Fang et al., 2018; Tzeng et al., 2016; Bousmalis et al., 2018; James et al., 2019). A final approach is to learn perception and control separately so that the real world perception system is only trained with real world images (Zhang et al., 2016; Devin et al., 2017). The problem of overcoming a reality gap in the agent's observations of the world is orthogonal to the problem of differing dynamics that we study.

4 Grounded simulation learning

In this section we introduce the *grounded simulation learning* (GSL) framework as presented by Farchy et al. (2013). Our main contribution is a novel algorithm that instantiates this general framework. GSL allows reinforcement learning in simulation to succeed by using trajectories from $\mathcal{M}$ to first modify $\mathcal{M}_{\texttt{sim}}$ such that the modified $\mathcal{M}_{\texttt{sim}}$ is a higher fidelity model of $\mathcal{M}$. The process of making the simulator more like the real world is referred to as *grounding*.

The GSL framework assumes the following:

1. There is an imperfect simulator MDP, $\mathcal{M}_{\texttt{sim}}$, that models the MDP environment of interest, $\mathcal{M}$. Furthermore, $\mathcal{M}_{\texttt{sim}}$ must be *modifiable*. In this article, we formalize modifiable as meaning that the simulator has parameterized transition probabilities $P_{\boldsymbol{\phi}}(\cdot|s,a) := P_{\texttt{sim}}(\cdot|s,a;\boldsymbol{\phi})$ where the vector $\boldsymbol{\phi}$ can be changed to produce, in effect, a different simulator.

2. There is a policy improvement algorithm, `optimize`, that searches for π which increase $v(\pi, \mathcal{M}_{\texttt{sim}})$. The `optimize` routine returns a set of candidate policies, Π to evaluate in $\mathcal{M}$.

We formalize the notion of grounding as minimizing a similarity metric between the real world trajectories and the trajectory distribution of the simulation. Let $d(h, \Pr_{\texttt{sim}}(\cdot|\pi;\boldsymbol{\phi}))$ be a score for the likelihood of a given trajectory in the simulator parameterized by $\boldsymbol{\phi}$. Given a dataset of trajectories, $\mathcal{D}_{\texttt{real}} := \{h_i\}_{i=1}^m$, collected by running a policy, π, in $\mathcal{M}$, simulator grounding of $\mathcal{M}_{\texttt{sim}}$ amounts to finding $\boldsymbol{\phi}^\star$ such that:

$$\boldsymbol{\phi}^\star = \arg\max_{\boldsymbol{\phi}} \sum_{h \in D_{\texttt{real}}} d\big(h, \Pr_{\texttt{sim}}(\cdot|\pi;\boldsymbol{\phi})\big). \tag{1}$$

For instance, if $d(h, \Pr_{\texttt{sim}}(\cdot|\pi;\boldsymbol{\phi})) := \log \Pr_{\texttt{sim}}(h|\pi;\boldsymbol{\phi})$ then $\boldsymbol{\phi}^\star$ maximizes the negative log-likelihood or equivalently the empirical Kullback-Leibler divergence between $\Pr(\cdot|\pi, \mathcal{M})$ and $\Pr_{\texttt{sim}}(\cdot|\pi, \boldsymbol{\phi}^\star)$.

Intuitively, Eq. (1) is solved by making the real world trajectories under π more likely when running π in the simulator. Though exactly solving Eq. (1) may be intractable, if we can make real world trajectories *more* likely in the simulator then the simulator will be better for policy optimization. Assuming a mechanism for optimizing (1), the GSL framework is as follows:

1. Execute an initial policy, π_0, in the real world to collect a data set of trajectories, $\mathcal{D}_{\texttt{real}} = \{h_j\}_{j=1}^m$.
2. Optimize (1) to find $\boldsymbol{\phi}^\star$ that makes $\Pr(H = h|\pi_0, \mathcal{M}_{\texttt{sim}})$ closer to $\Pr(H = h|\pi_0, \mathcal{M})$ for all $h \in \mathcal{D}_{\texttt{real}}$.
3. Use `optimize` to find a set of candidate policies Π that improve $v(\cdot, \mathcal{M}_{\texttt{sim}})$ in the modified simulation.
4. Evaluate each proposed $\pi_c \in \Pi$ in $\mathcal{M}$ and return the policy:

$$\pi_1 := \arg\max_{\pi_c \in \Pi} v(\pi_c, M).$$

GSL can be applied iteratively with π_1 being used to collect more trajectories to ground the simulator again before learning π_2. The re-grounding step is necessary since changes to π result in changes to the distribution of trajectories that the agent observes. When the distribution changes, a simulator that has been modified with data from the trajectory distribution of π_0 may be a poor model under the trajectory distribution of π_1. The entire GSL framework is illustrated in Fig. 1.

5 The grounded action transformation algorithm

We now introduce the main contribution of this article—a novel GSL algorithm called the *grounded action transformation* (GAT) algorithm. GAT instantiates the GSL framework by introducing a specific implementation of the grounding step (Step 2) of the GSL framework. The main idea behind GAT is to augment the simulator with a differentiable *action transformation function*, g, which transforms the agent's simulated action into an action which—when taken in simulation—produces the same transition that would have occurred in the physical system. The function, g, is represented with a parameterized function

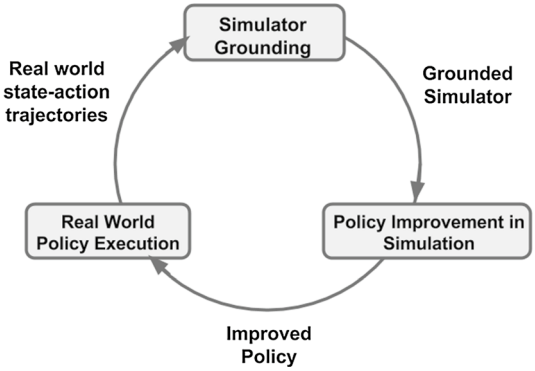


Fig. 1 Diagram of the grounded simulation learning framework

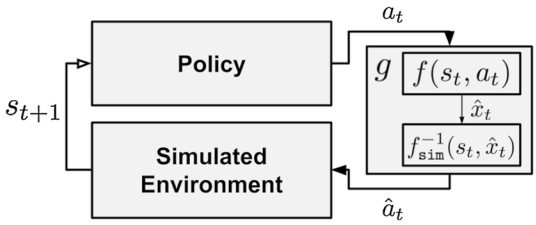


Fig. 2 The augmented simulator which can be grounded to the real world with supervised learning. The policy computes an action that is then passed to the action grounding module. This module first predicts the values for the state variables of interest if the action had been taken in the real world. The module then uses an inverse dynamics model, $f^{-1}_{\texttt{sim}}$, to compute the action that produces the same effect in simulation. Finally, the policy's action is replaced with the predicted action and this modified action is passed to the simulator

approximator whose parameters serve as $\boldsymbol{\phi}$ for the augmented simulator in the GSL framework. We leave open the GAT instantiation of the other GSL steps (data collection, policy optimization, and final policy evaluation). The main contribution of GAT is a novel method to ground the simulator.

The GAT algorithm learns two functions: f which predicts the effects of actions in $\mathcal{M}$ and $f^{-1}_{\texttt{sim}}$, which predicts the action needed in simulation to reproduce the desired effects. Let $\mathbf{x}$ be a subset of the components of state $\mathbf{s}$ and let $\mathcal{X}$ be the set of all possible values for $\mathbf{x}$. We refer to the components of $\mathbf{x}$ as the *state variables of interest*. We define GAT as grounding a subset of the state components to allow users to inject domain knowledge into the grounding process if they know what components are most important to model correctly; a user can always opt to include all components of the state as state variables of interest if they lack such domain knowledge. Formally, the function $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{X}$ is a forward model that predicts the effect on the state variables of interest given an action chosen in a particular state in $\mathcal{M}$. The function $f^{-1}_{\texttt{sim}} : \mathcal{S} \times \mathcal{X} \rightarrow \mathcal{A}$ is an inverse model that predicts the action that causes a particular effect on the state variables of interest given the current state in simulation. The overall action transformation function $g : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{A}$ is specified as $g(\mathbf{s}, \mathbf{a}) := f^{-1}_{\texttt{sim}}(\mathbf{s}, f(\mathbf{s}, \mathbf{a}))$. When the agent is in state $\mathbf{s}_t$ in the simulator and takes action $\mathbf{a}_t$, the augmented simulator replaces $\mathbf{a}_t$ with $g(\mathbf{s}_t, \mathbf{a}_t)$ and the simulator returns $\mathbf{s}_{t+1}$ where the $\mathbf{x}_{t+1}$ components of $\mathbf{s}_{t+1}$ are closer in value to what would be observed in $\mathcal{M}$ had $\mathbf{a}_t$ been taken there. Figure 2 illustrates the augmented simulator.

GAT learns the functions f and $f_{\texttt{sim}}^{-1}$ with supervised learning. The function f is learned by collecting a small number of real world trajectories and then constructing a supervised learning dataset $\{(\mathbf{s}_i, \mathbf{a}_i)\} \rightarrow \{\mathbf{x}'_i\}$. Similarly, the function $f_{\texttt{sim}}^{-1}$ is learned by collecting simulated trajectories and then constructing a supervised learning dataset $\{(\mathbf{s}_i, \mathbf{x}'_i)\} \rightarrow \{\mathbf{a}_i\}$. This pair of supervised learning problems can be solved by a variety of techniques. In our experiments we use either neural networks or linear models trained with gradient descent on a squared error loss. Pseudocode for the full GAT algorithm is given in Algorithm 1.

Algorithm 1 Grounded Action Transformation (GAT). **Input:** An initial policy, π_0, the environment, $\mathcal{M}$, a simulator, $\mathcal{M}_{\texttt{sim}}$, and a policy improvement method, `optimize`. The function `rollout`$(\texttt{Env}, \pi, \texttt{m})$ executes m trajectories with π in the provided environment, `Env`, and returns the observed state transition data. The functions `trainForwardModel` and `trainInverseModel` estimate models of the forward and inverse dynamics respectively given a dataset of trajectories. The function `optimize` takes the simulator, an initial policy, and the grounding function, g, and runs an RL algorithm that finds policies that improve on the initial policy in the grounded simulator.

$i \leftarrow 0$
repeat
  $\mathcal{D}_{\texttt{real}} \leftarrow \texttt{Rollout}(\mathcal{M}, \pi_i, m)$
  $\mathcal{D}_{\texttt{sim}} \leftarrow \texttt{Rollout}(\mathcal{M}_{\texttt{sim}}, \pi_i, m)$
  $f \leftarrow \texttt{trainForwardModel}(\mathcal{D}_{\texttt{real}})$
  $f_{\texttt{sim}}^{-1} \leftarrow \texttt{trainInverseModel}(\mathcal{D}_{\texttt{sim}})$
  $g(s, a) \leftarrow f_{\texttt{sim}}^{-1}(s, f(s, a))$
  $\mathit{\Pi} \leftarrow \texttt{optimize}(\mathcal{M}_{\texttt{sim}}, \pi_i, g)$
  $i \leftarrow i + 1$
  $\pi_i \leftarrow \text{argmax}_{\pi \in \mathit{\Pi}}\, v(\pi)$
until $v(\pi_i) < v(\pi_{i-1})$ // No improvement in real world performance.

Return $\text{argmax}_i\, v(\pi_i)$

Because we take a data-driven approach to simulator modification, the result is not necessarily a globally more accurate simulator for the real world. Our only goal is that the simulator is more realistic for trajectories *sampled with the grounding policy*. If we can achieve this goal, then we can locally improve the policy without any additional real world data. A simulator that is more accurate globally may provide a better starting point for GAT, however, by focusing on simulator modification local to the grounding policy we can still obtain policy improvement in low fidelity simulators.

We also note that GAT minimizes the error between the immediate state transitions of $\mathcal{M}_{\texttt{sim}}$ and those of $\mathcal{M}$. Another possible objective would be to observe the difference between trajectories in $\mathcal{M}$ and $\mathcal{M}_{\texttt{sim}}$ and ground the simulator to minimize the total error over a trajectory. Such an objective could lead to an action modification function g that accepts short-term error if it reduces the error over the entire trajectory, however, it would require the simulator dynamics to be differentiable. As it is unclear how to select the modified actions that minimize multi-step error, we accept minimizing the one-step error as a good proxy for minimizing our ultimate objective which is that the current policy π produces similar trajectories in both $\mathcal{M}$ and $\mathcal{M}_{\texttt{sim}}$. The specific choice of g used by GAT allows GAT to learn the actions that minimize the one-step error in simulated and real world transitions.

5.1 Modifying actions vs. modifying parameters

Before presenting an empirical evaluation of GAT, we discuss the motivation for modifying actions instead of internal simulator parameters. Our main motivation for modifying the agent's simulated action is that we can then treat the simulator as a black box. While physics-based simulators typically have a large number of parameters determining the physics of the simulated environment (e.g., friction coefficients, gravitational values) these parameters are not necessarily amenable to numerical optimization of Eq. (1). First, just because a simulator has such parameters does not mean that they're exposed to the user or can be modified without additional software engineering. On the other hand, when applying RL, it is reasonable to assume that a user has access to the actions output by the policy and could thus include an action transformation to ground the simulator. Second, even if changing physics parameters is straightforward, it may be computationally or manually intensive to determine how to change a parameter to make the simulator produce trajectories closer to the ones we observe in the real world. In contrast, action modification with GAT allows us to transform simulator modification into a supervised learning problem.

In this article we focus on the blackbox setting where we are unable to change the simulator's internal parameters. However, if these parameters are exposed to the user then there may be settings where correctly identifying the real world parameters may provide more reliable transfer than action modification. A characterization of the settings where one approach is preferable to the other is an interesting direction for future research.

6 GAT empirical study

We now present an empirical study of applying the GAT algorithm for reinforcement learning with simulated data. Our experiments are designed to answer the following questions:

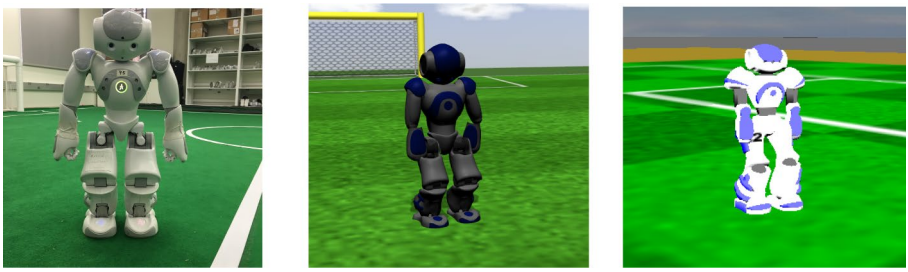
1. Does grounding a simulation with GAT allow skills learned in simulation to transfer to the real world?
2. Does GAT make the simulated robot's actions have similar effects to those they would have in the real world.

To answer these questions we apply GAT on three tasks with the simulated and physical NAO robot. Though our focus is on sim-to-real transfer, we include two experiments in a sim-to-sim setting where we use one simulator as a surrogate for the real world. These experiments allow us to run a larger number of experimental trials than would be practical in the tasks using a physical robot. We first give a general description of the empirical setup. We then proceed to describe each task and the empirical results observed.

6.1 General NAO task description

All empirical tasks use either a simulated or physical Softbank NAO robot.[2] The NAO is a humanoid robot with 25 degrees of freedom (see Fig. 3a). Though the NAO has 25 degrees of freedom, we restrict ourselves to observing and controlling 15 of them (we ignore joints

[2] https://www.ald.softbankrobotics.com/en.



(a) A Softbank NAO Robot **(b)** NAO in Gazebo **(c)** NAO in SimSpark

Fig. 3 The three robotic environments used here. The Softbank NAO is our target physical robot. The NAO is simulated in the Gazebo and SimSpark simulators. Gazebo is a higher fidelity simulator which we also use as a surrogate for the real world in an empirical comparison of grounded action transformation (GAT) to baseline methods

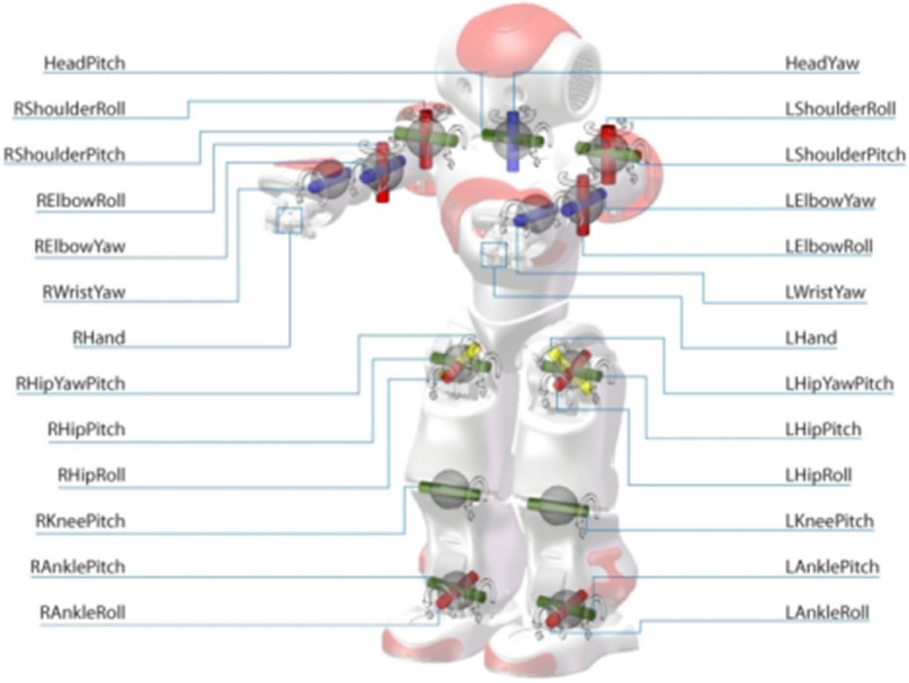


Fig. 4 Diagram of the Softbank NAO robot with joints (degrees of freedom) labeled. Each joint has a sensor that reads the current angular position of the joint and can be controlled by providing a desired angular position for the joint. In this work, we ignore the HeadYaw, HeadPitch, left and right ElbowRoll, left and right ElbowYaw, left and right WristYaw, and left and right Hand joints. There is also no need to control the right HipYawPitch joint as, in reality, this degree of freedom is controlled by the movement of the left HipYawPitch Joint. This image was downloaded from: http://doc.aldebaran.com/2-8/family/nao_technical/lola/actuator_sensor_names.html

that are less important for our experimental tasks—joints in the head, hands, and elbows). We will refer to the degrees of freedom as the joints of the robot. Figure 4 shows a diagram of the NAO and its different joints.

We define the state variables of interest to be the angular position of each of the robot's joints. In addition to angular position, the robot's state consists of joint angular velocities and other task-dependent variables. The robot's actions are desired joint angular positions which are implemented at a lower software level using PID control. There is a one-to-one correspondence between components of the robot's action and the state variables of interest.

In all tasks our implementation of GAT uses a history of the joint positions and desired joint positions as an estimate of the NAO's state to input into the forward and inverse models. Instead of directly predicting $\mathbf{x}_{t+1}$, the forward model, f, is trained to predict the change in $\mathbf{x}_t$ after taking $\mathbf{a}_t$. The inverse model $f^{-1}_{\texttt{sim}}$ takes the current $\mathbf{x}_t$ and a desired change at $\mathbf{x}_{t+1}$ and outputs the action needed to cause this change. Since both the state variables of interest and actions have angular units, we train both f and $f^{-1}_{\texttt{sim}}$ to output the sine and cosine of each output angle. From these values we can recover the predicted output with the arctan function. Since $f^{-1}_{\texttt{sim}}$ and f are trained with supervised learning, they may make small errors when used to change the agent's actions (Ross et al., 2011). Since small errors may make the output of g not smooth from timestep to timestep, we sometimes find it useful to use a smoothing parameter, α, to ensure stable motions. The action transformation function (Algorithm 1, line 7) is then defined as:

$$g(\mathbf{s}, \mathbf{a}) := \alpha f^{-1}_{\texttt{sim}}(\mathbf{s}, f(s, \mathbf{a})) + (1 - \alpha)\mathbf{a}.$$

In our experiments involving bipedal walking, we set α as high as possible subject to the robot remaining stable in simulation when executing π_0. In all other experiments, we use $\alpha = 1.0$.

We consider two simulators in this work: the Simspark[3] Soccer Simulator used in the annual RoboCup 3D Simulated Soccer competition and the Gazebo simulator from the Open Source Robotics Foundation.[4] SimSpark enables fast simulation but is a lower fidelity model of the real world. Gazebo enables relatively high fidelity simulation with an additional computational cost. The NAO model in both of these simulations is shown in Fig. 3a.

Across all tasks we use the covariance matrix adaptation evolutionary strategies (CMA-ES) algorithm (Hansen et al., 2003) for the policy optimization routine. CMA-ES is a stochastic search algorithm that updates a *population* of candidate policies over a set number of *generations*. At each generation, CMA-ES samples a population of policy parameter values from a Gaussian distribution. It then uses the evaluation of each candidate policy in simulation to update the sampling distribution for the population at the next generation. CMA-ES has been found to be very effective at optimizing robot skills in simulation (Urieli et al., 2011). In all experiments we use a population size of 150 candidate policies at each generation as we were able to submit up to 150 parallel policy evaluations at a time on the University of Texas Computer Science distributed computing cluster.

With the exception of the final experiment in this section, we run a single iteration of GAT per experimental setting. A single iteration allows us to keep the initial policy fixed so that we have a more controlled measure of the efficacy of simulator grounding. In all cases

[3] http://simspark.sourceforge.net.
[4] http://gazebosim.org.

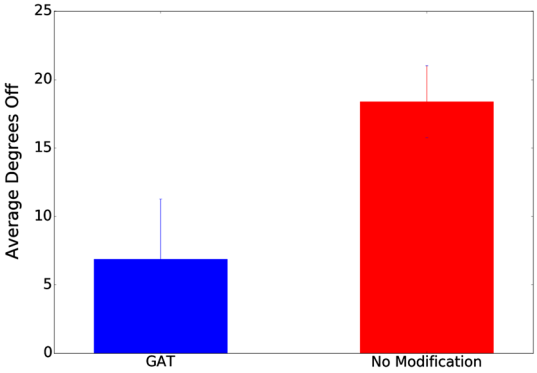


Fig. 5 Mean performance of best policies found on the Arm Control task. We run 10 experimental trials using GAT and 10 experimental trials directly transferring from $\mathcal{M}_{\texttt{sim}}$ to $\mathcal{M}$ ("No Modification"). The vertical axis gives the average distance to the target position during a trajectory (lower is better). Error bars are for a 95% confidence interval

we select the architectures of the forward and inverse dynamics models via optimizing a least-squares loss on a held-out set of transitions. These models are trained with stochastic gradient descent using the Adam optimizer (Kingma & Ba, 2014).

6.2 Learning arm control

Our first task requires the NAO to learn to raise its arms from its sides to a goal position, $\mathbf{p}^\star$ which is defined to be halfway to horizontal (lift 45 degrees). We call this task the "Arm Control" task. In this task, the robot's policy only controls the two shoulder joints responsible for raising and lowering the arms. The angular position of these joints are the state variables of interest, $\mathbf{x}$. The policy is a linear mapping from $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ to the action $\mathbf{a}_t$:

$$\pi(\mathbf{x}_t, \mathbf{x}_{t-1}) = \mathbf{w} \cdot (\mathbf{x}_t, \mathbf{x}_{t-1}) + \mathbf{b}$$

where $\mathbf{w}$ and $\mathbf{b}$ are learnable parameters. At time t, the agent receives reward:

$$r(\mathbf{x}_t) = \frac{1}{|\mathbf{x}_t - \mathbf{p}^\star|_2^2}$$

and the episode terminates after 200 steps or when either of the robot's arms raise higher than 45 degrees. The optimal policy is to move as close as possible to 45 degrees without lifting higher.

We apply GAT for sim-to-sim transfer from Simspark ($\mathcal{M}_{\texttt{sim}}$) to Gazebo ($\mathcal{M}$ – effectively treating Gazebo as the real world). We represent f and $f^{-1}_{\texttt{sim}}$ with linear functions. To train f, we collect 50 trajectories in $\mathcal{M}$ and train $f^{-1}_{\texttt{sim}}$ with 50 trajectories from $\mathcal{M}_{\texttt{sim}}$.

On this task our baseline is learning without simulator modification. For each method (GAT and "No Modification"), we run 10 experimental trials where each trial consists of running 50 generations of CMA-ES and taking the best performing candidate policy from each generation and evaluating it in $\mathcal{M}$. Our main point of comparison is which method finds a policy that allows the robot to move its arms closer to the target position (higher $v(\pi, \mathcal{M})$).

Figure 5 shows the mean distance from the target position for the final policy learned in simulation either with GAT or with "No Modification." Results show that GAT is able to overcome the reality gap and results in policies that reduce error in final arm position.

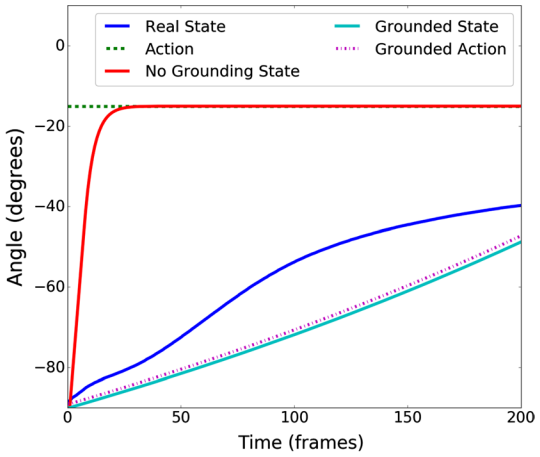


Fig. 6 Visualization of the robot's LeftShoulderPitch joint position in $\mathcal{M}$, $\mathcal{M}_{\texttt{sim}}$, and $\mathcal{M}_{\texttt{sim}}$ after applying GAT. The horizontal axis is time in frames (50 frames per second). The vertical axis has units of angles which is the unit for both the plotted actions and states. Trajectories were generated in each environment with a policy that sets a constant desired position of −15 degrees ("Action"). "Real State" shows the LeftShoulderPitch position in $\mathcal{M}$, "No Grounding State" shows position in $\mathcal{M}_{\texttt{sim}}$, and "Grounded State" shows position in the grounded $\mathcal{M}_{\texttt{sim}}$. "Grounded Action" shows the action that the GAT action modification function takes in place of "Action"

We also visualize the effect of the action modification function, g, in the simulator. Figure 6 shows how the robot's LeftShoulderPitch joint moves in $\mathcal{M}$, $\mathcal{M}_{\texttt{sim}}$, and the grounded $\mathcal{M}_{\texttt{sim}}$ when a constant action of −15 degrees is applied. In $\mathcal{M}_{\texttt{sim}}$ the position of the LeftShoulderPitch responds immediately to the command while in $\mathcal{M}$ the position changes much more slowly. In Simspark, the shoulder joints are more responsive to commands and thus the robot needs to learn it must take weaker actions to prevent overshooting the target. In Gazebo, the joints are less responsive to the actions and the same policy fails to get the arms close to the target. After applying GAT, the position changes much slower in simulation as the action modification function reduces the magnitude of the desired change. This visualization helps answer our second empirical question as to whether or not action modification makes the simulator behave more like reality.

6.3 Linear walk policy optimization

Our second task is walking forward with a linear control policy on the physical robot. The state variables of interest are 10 joints in the robot's legs (ignoring the left HipYawPitch joint) and the 4 joints controlling its shoulders. The actions are desired angular positions for all 15 of these joints.

The policy inputs are the gyroscope that measures forward-backward angular velocity, y, and the gyroscope that measures side-to-side angular velocity, x. We also provide as input an open-loop sine wave. The sine wave encodes prior knowledge that a successful walking policy will repeat actions periodically. The final form of the policy is:

$$\pi(\langle x, y, \sin(c \cdot t)\rangle) = \mathbf{w} \cdot \langle x, y, \sin(c \cdot t)\rangle + \mathbf{b}$$

where c is a learnable scalar that controls the walking step frequency. The policy outputs only commands for the left side of the robot's body and the commands for the right side are obtained by reflecting these commands around a learned value. That is, for each joint, j, on the left side of the robot's body we learn a parameter ψ_j and obtain the action for the right side of the robot's body by reflecting the policy's output for j across ψ_j. This representation is equivalent to expressing the policy for the right side of the robot's body as:

$$\pi_r(\langle x, y, \sin(c \cdot t)\rangle) = \boldsymbol{\psi} - (\mathbf{w} \cdot \langle x, y, \sin(c \cdot t)\rangle + \mathbf{b} - \boldsymbol{\psi}).$$

In our experiments, instead of optimizing a separate $\boldsymbol{\psi}$ vector, we clamp $\boldsymbol{\psi}$ to be equal to the bias, $\mathbf{b}$.

We define the reward as a function of the distance the robot has travelled at the final time-step. Let $\Delta(s_t, s_0)$ be the robot's forward change in position between state s_t and state s_0 and let $\mathbb{I}(s_t)$ take value 1 if the robot has fallen over in state s_t and 0 otherwise. In simulation:

$$r(s_t, a_t) := \begin{cases} 0 & t < l - 1 \\ \Delta(s_t, s_0) - 25 \cdot \mathbb{I}(s_t) & t = l \end{cases}.$$

where the penalty of -25 discourages CMA-ES from proposing policies that obtain high forward displacement through potentially unsafe actions for the physical robot. For example, CMA-ES might find a policy that throws itself forward, obtaining high reward but risking damage on the physical robot. The penalty does *not* guarantee that the best simulation policies will be stable in the real world but it at least encourages them to be stable in simulation. On the physical robot we only measure forward distance travelled; if the robot falls we count the distance travelled as zero:

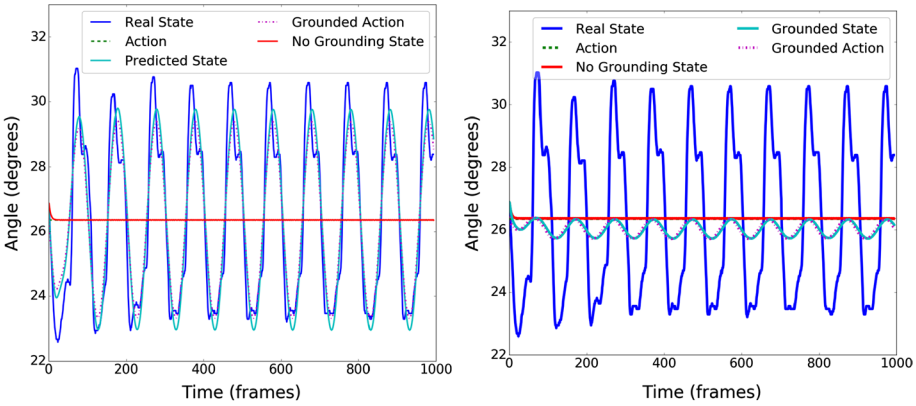
$$r(s_t, a_t) := \begin{cases} 0 & t < l - 1 \\ \Delta(s_t, s_0) \cdot (1 - \mathbb{I}(s_t)) & t = l \end{cases}.$$

We apply GAT for sim-to-real transfer from Simspark to the physical NAO. We learn f and $f_{\texttt{sim}}^{-1}$ with linear regression. To train f we collect 10 trajectories in $\mathcal{M}$ and train $f_{\texttt{sim}}^{-1}$ with 50 trajectories from $\mathcal{M}_{\texttt{sim}}$. We chose 10 trajectories for $\mathcal{M}$ because after 10 the robot's motors may begin to heat up which changes the dynamics of the joints.

In the Linear Policy Walking task we measure performance based on how far forward the robot walks. The initial policy fails to move the robot forward at all—though it is executing a walking controller, its feet never break the friction of the carpet and so it remains at the starting position. We run five trials of learning with simulator modification and five trials without. On average learning in simulation with GAT resulted in the robot moving 4.95 cm forward while without simulator modification the robot only moved 1.3 cm on average.

Across the five trials *without* modification, two trials fail to find any improvement. The remaining three only find improvement in the first generation of CMA-ES—before CMA-ES has been able to begin exploiting inaccuracies in the simulation. In contrast, all trials *with* simulator modification find improving policies and improvement comes in later learning generations (on average generation 3 is the best).

We also plot example trajectories to see how the modified and unmodified simulations compare to reality. Instead of plotting all state and action variables, we only plot the state variable representing the robot's right AnklePitch joint and the action that specifies a desired position for this joint. This joint was chosen because the main failure of policies



(a) Predicted trajectory **(b)** Actual trajectory

Fig. 7 Visualization of the robot's right AnklePitch joint during the Linear Policy Walking task. Both subfigures show the position trajectory for $\mathcal{M}$ (denoted "Real State") and $\mathcal{M}_{\texttt{sim}}$ ("No Grounding State"). They also both show the action though it is covered by the "No Grounding State" curve. **a** shows the GAT forward model's prediction of position given the same action sequence. **b** shows the actual position when acting in the modified simulation

learned without simulator modification is that the robot's feet never break the friction of the carpet. We hypothesize that learning to properly move the ankles may be important for a policy to cross the reality gap and succeed in the real world.

Figure 7a shows the prediction of joint position for the learned forward model, f, as well as the joint position in the real world and simulation. The "Predicted State" curve is generated by using f as a simulator of how the joint position changes in response to the actions.[5] Figure 7a shows that in the real world the right AnklePitch joint oscillates around the desired angular position as given by the robot's action. The forward model f predicts this oscillation while the simulator models the joint position as static.

Figure 7b shows the actual real world and simulated trajectories, both for the modified and unmodified simulators. Though the modified simulator still fails to capture all of the real world oscillation, it does so more than no modification. Learning in a simulator that more accurately models this motion leads to policies that are able to lift the robot's legs enough to walk. This qualitative results also shows how action modification can be an effective strategy for simulator grounding.

6.4 Sim-to-sim walk engine policy optimization

In this section, we evaluate GAT on the task of bipedal robot walking with a state-of-the-art walk controller for the NAO robot. The initial policy is the open source University of New South Wales (UNSW) walk engine developed for RoboCup Standard Platform League (SPL) competitions (Ashar et al., 2015; Hall et al., 2016). This walk engine is a software module designed for the NAO robot that takes in the robot's proprioceptive and inertial sensors and outputs desired positions for the robot's joints; we refer the reader to Ashar et al. (2015)

[5] Note that f would *not* suffice for policy improvement as it only models how the joint positions change and *not* the effect of these changes on walk velocity.

Table 1 The initial parameter values found in the open source release of the UNSW walk engine

Parameter name	Parameter value
Center of mass offset	0.01
Base walk period	0.23
Walk hip height	0.23
Max forward	0.3
Max left step	0.2
Max turn	0.87
Max forward change	0.15
Max left change	0.2
Max turn change	0.8
Base leg lift	0.012
Arm swing	6.0
Pendulum height	300.0
Forward extra foot height	0.01
Left extra foot height	0.02
Start lift divisor	3.5

Some of these values were explicit parameters in the open source release; others were hard-coded constants that we chose to allow CMA-ES to modify during policy optimization

for full details of the initial policy's implementation. This walk controller has been used by at least one team in the 2014, 2015, 2016, 2017, 2018, 2019 RoboCup Standard Platform League (SPL) championship games in which teams of five NAOs compete in soccer matches. To the best of our knowledge, it is the fastest open source walk available for the NAO. We first present a sim-to-sim evaluation of GAT using Gazebo as a surrogate for the real world. Performing a sim-to-sim evaluation allows us to evaluate GAT and baselines with more trials than would be possible to run on the physical robot. In the next section, we apply GAT to optimize the UNSW walk engine the physical robot.

The UNSW walk engine has 15 parameters that determine features of the walk (see Table 1 for a full list of these parameters). The values of the parameters from the open source release constitute the parameterization of the initial policy π_0. Hengst (2014) describes the UNSW walk controller in more detail. For this task, $v(\pi, \mathcal{M})$ is the average forward walk velocity while executing π. In simulation a trajectory terminates after a fixed time interval (7.5 seconds in SimSpark and 10 seconds in Gazebo) or when the robot falls. For policy improvement in simulation, we apply CMA-ES for 10 generations with a population size of 150 candidate policies evaluated in each generation.

We implement GAT with two two-hidden-layer neural networks—one for f and one for $f^{-1}_{\texttt{sim}}$. Each function is a neural network with 200 hidden units in the first layer and 180 hidden units in the second.

As baselines, we evaluate the effectiveness of GAT compared to learning with no grounding and grounding $\mathcal{M}_{\texttt{sim}}$ by adding Gaussian noise to the robot's actions. Adding an "envelope" of noise has been used before to minimize simulation bias by preventing the policy improvement algorithm from overfitting to the simulator's dynamics (Jakobi et al., 1995). We refer to this baseline as ANE for Action Noise Envelope. We hypothesize that GAT is modifying simulation in a more effective way than just forcing learning to be robust to perturbation and will thus obtain a higher level of performance.

Table 2 This table compares the grounded action transformation algorithm (GAT) with baseline approaches for transferring learning between SimSpark and Gazebo

Method	% Improve	Transfer failures	Best iteration
No Ground	11.094	7	1.33
ANE	18.93	5	6.6
GAT	**22.48**	**1**	2.67

The first column displays the average maximum improvement found by each method after the first policy update made by CMA-ES. The second column is the number of times a method failed to find a stable walk. The third column gives the average generation of CMA-ES when the best policy was found. No Ground refers to learning done in the unmodified simulator. Bold values indicate the best performance for each of the first two columns

For GAT we collect 50 trajectories of robot experience to train f and 50 trajectories of simulated experience to train $f^{-1}_{\texttt{sim}}$. For each method, we run 10 generations of the CMA-ES algorithm with population size of 150 and each member of the population evaluated in simulation with 20 trajectories. Overall, the CMA-ES optimization requires 30,000 simulated trajectories for each experimental trial. We run 10 total experimental trials for each method.

Table 2 gives the average improvement in stable walk policies for each method and the number of trials in which a method failed to produce a stable improvement. Results show that GAT maximizes policy improvement while minimizing failure to transfer when transferring from a low-fidelity to high-fidelity simulator. ANE improves upon no grounding in both improvement and number of iterations without improvement. Adding noise to the simulator encourages CMA-ES to propose robust policies which are more likely to be stable. However, GAT further improves over ANE—demonstrating that action transformations are grounding the simulator in a more effective way than simply injecting noise.

Table 2 also shows that on average, GAT finds an improved policy within the first few generations after grounding. The grounding done by GAT is inherently local to the trajectory distribution of π_{θ_0}. Thus as π_θ changes, the action transformation function fails to produce a more realistic simulator. As policy improvement progresses, the best policies in each CMA-ES generation begin to over-fit to the dynamics of $\mathcal{M}_{\texttt{sim}}$. Without grounding over-fitting happens almost immediately and so when learning with no grounding finds an improvement it is also usually in an early generation of CMA-ES. ANE can mitigate over-fitting by emphasizing robust policies although it is limited in the improvement it finds compared to GAT.

6.5 Sim-to-real walk engine policy optimization

We now present our main empirical result—an application of GAT to optimizing a state-of-the-art walking controller for the NAO robot. All experimental details are the same as those used in the sim-to-sim evaluation except for the following changes. On the physical robot, a trajectory terminates once the robot has walked four meters ($\approx$ 20.5s with the initial policy) or falls. The data set $\mathcal{D}$ consists of 15 trajectories collected with π_0 on the physical NAO. To ensure the robot's motors stayed cool, we waited five minutes after collecting every five trajectories. For each iteration of GAT, we run 10 generations of the CMA-ES algorithm with a population size of 150. For each generation of CMA-ES we select $\arg\max\, v(\pi, M_{\texttt{sim}})$ and

Table 3 This table gives the maximum learned velocity and percent improvement for each method starting from π_0 (top row)

Method	Velocity (cm/s)	% Improve
π_0	19.52	0.0
GAT SimSpark π_1	26.27	34.58
GAT SimSpark π_2	27.97	43.27
GAT Gazebo π_1	26.89	37.76

evaluate it on the physical robot (resulting in 10 policies being evaluated on the physical robot). We evaluate each policy on the physical robot with five trajectories. If the robot falls in any trajectory the policy is considered unstable.

Table 3 gives the physical world walk velocity of policies learned in simulation with GAT. The physical robot walks at a velocity of 19.52 cm/s with π_0. GAT with SimSpark and GAT with Gazebo both improved walk velocity by over 30% in a single iteration. Policy improvement with CMA-ES required 30,000 trajectories per GSL iteration to find the 10 policies that were evaluated on the robot. In contrast the total number of trajectories executed on the physical robot is 65 (15 trajectories in $\mathcal{D}$ and 5 evaluations per $\pi_c \in \Pi$). This result demonstrates GAT can use sample-intensive simulation learning to optimize real world skills with a low number of trajectories on the physical robot.

Farchy et al. (2013) demonstrated the benefits of re-grounding (i.e., re-running the GSL framework from the best policy found) and further optimizing π. We reground the simulator with 15 trajectories collected with the best policy found by GAT with SimSpark and optimize for a further 10 generations of CMA-ES in the SimSpark simulation. The second iteration of GAT results in a walk, $\boldsymbol{\theta}_2$, which averages 27.97 cm/s for a total improvement of 43.27% over $\boldsymbol{\theta}_0$.[6] Overall, improving the UNSW walk by over 40% shows that GAT can learn walk policies that outperform the fastest known stable walk for the NAO robot.

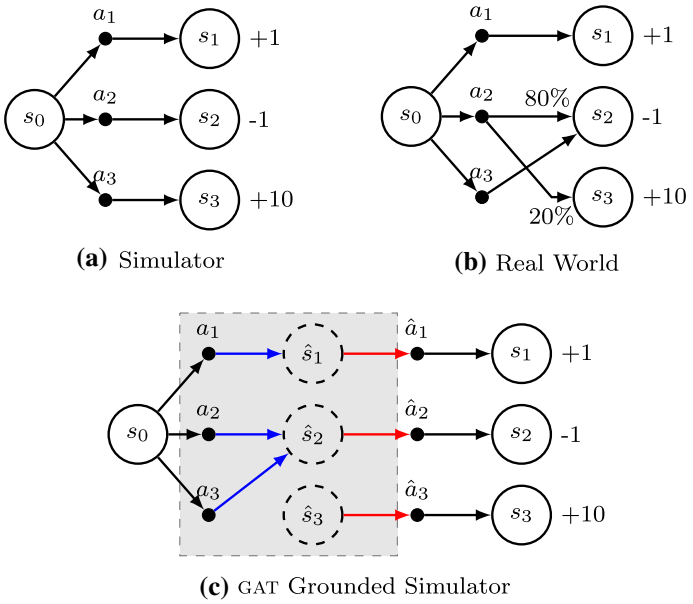
7 Stochastic GAT (SGAT)

The experiments described in Sect. 6 established that GAT can lead to successful sim-to-real transfer on a challenging task. This success naturally raises the question of under what conditions GAT will succeed, and, on the other hand, when it might fail. Towards answering this question, we observe that because GAT learns a deterministic forward model of the world, it may be limited when the real world state transitions are stochastic. We then introduce a generalization of GAT and demonstrate how it overcomes this limitation.

When the real world has stochastic transitions, GAT may be unable to ground the simulator in a way that leads to a good policy. To see this limitation, consider the toy example shown in Fig. 8. In Fig. 8, the optimal action in the simulator is a_3, and in the real world, it is a_2; however, in the GAT grounded simulator, the optimal action becomes a_1. Since GAT's forward model is deterministic, it predicts only the most likely next state, but other, less likely transitions are also important when computing an action's value.

To address real world stochasticity, we introduce a generalization of GAT—Stochastic Grounded Action Transformation (SGAT)—which learns a stochastic model of the

[6] A video of the learned walk policies is available at https://www.cs.utexas.edu/users/AustinVilla/?p=research/real_and_sim_walk_learning.



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

(c) GAT Grounded Simulator

Fig. 8 A toy example where GAT may fail to ground the simulator for learning. The gray box depicts the grounding step with blue arrows representing the forward model and red arrows representing the inverse dynamics model. When the real world has stochastic transitions, the GAT forward model only captures the most likely next state. GAT may fail here, since the optimal action in the grounded simulator (a_3) is sub-optimal in the real environment

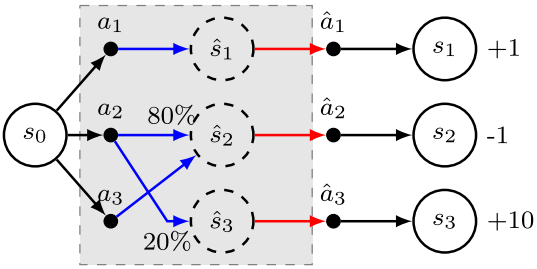


Fig. 9 The SGAT algorithm applied to the toy example in Fig. 8. In the SGAT Grounded Simulator, the transitions match the real environment (Fig. 8b)

forward dynamics. In other words, the learned forward model, f_{real}, predicts a distribution over next states, a potential next state is sampled from this distribution, and then the sampled state is used with f_{sim}^{-1} instead of always taking the most likely next state. The grounding function learned by SGAT is given by:

$$g(s,a) = f_{\text{sim}}^{-1}(s,S') \quad S' \sim f(s,a)$$

where $f(s, a)$ now gives a distribution over next states instead of the single most likely next state. The sampling operation within the action transformer makes the overall action transformation process stochastic. Figure 9 illustrates the simulator from the example in Fig. 8

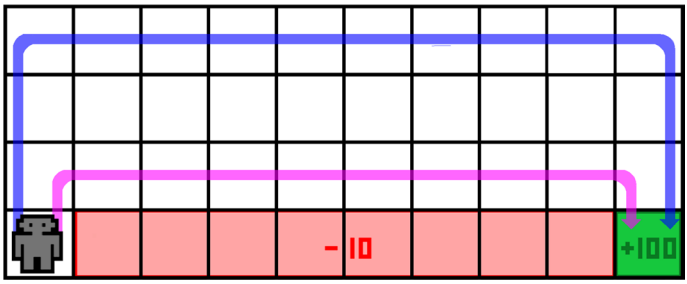


Fig. 10 The agent starts in the bottom left and must reach the goal in the bottom right. Stepping into the red region penalizes the robot and ends the episode. The purple path is the most direct, but the blue path is safer when the transitions are stochastic (Color figure online)

now grounded using SGAT. Since the forward model accounts for stochasticity in the real world, the actions in the grounded simulator have the same effect as in the real world.

An implementation of GAT can be extended to an implementation of SGAT by replacing the predicted next state output of f with predicted parameters of the next state distribution. Let $p(s_{t+1}|s_t, a_t)$ denote the probability of s_{t+1} under the distribution given by $f(s_t, a_t)$. We can fit the stochastic forward model to the observed real world data by minimizing a negative log likelihood loss $\mathcal{L} = -\log p(s_{t+1}|s_t, a_t)$ on the observed real world transition (s_t, a_t, s_{t+1}). For example, in continuous state and action domains, f could output the mean and covariance of a Gaussian distribution.

SGAT generalizes GAT as GAT can be seen as a variant of SGAT that always samples the most likely real world state given the current state and action. We next present an empirical study that shows that this generalization is crucial for real world domains with high stochasticity.

8 Stochastic GAT empirical study

This section reports on an empirical study of transfer from simulation with SGAT compared to GAT. We begin with a toy RL domain and progress to sim-to-real transfer of a bipedal walking controller for a NAO robot on bumpy carpet. This additional empirical study is designed to answer the questions:

1. Does GAT perform worse when real world stochasticticy is increased?
2. Can SGAT successfully ground simulation even when the real world is stochastic?

Our empirical results show the benefit of modelling stochasticity when grounding a simulator for transfer to a stochastic real world environment.

8.1 Cliff walking

We first verify the benefit of SGAT using a classical reinforcement learning domain, the Cliff Walking grid world (Sutton & Barto, 1998) shown in Fig. 10. In this domain, an agent must navigate around a cliff to reach a goal. The agent can move up, down, left, or right. If it tries to move into a wall, the action has no effect. The episode terminates when the agent

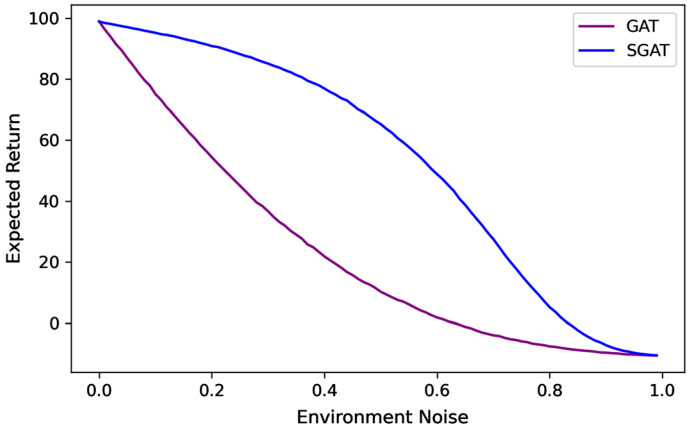


Fig. 11 The y-axis is the average performance of a policy evaluated on the "real" domain. The x-axis is the chance at each time step for the transition to be random. SGAT outperforms GAT for any noise value. Error bars not shown since standard error is smaller than 1 pixel

either reaches the goal (reward of +100) or falls off the cliff (reward of −10). There is also a small time penalty (−0.1 per time step), so the agent is incentivized to find the shortest path. There is no discounting, so the agent's objective is to maximize the sum of rewards over an episode. We use policy iteration (Sutton & Barto, 1998) for the `optimize` routine in simulation.

We make Cliff Walking a sim-to-sim transfer problem by treating a variant of the domain with deterministic transitions as the simulator and a variant of the domain with stochastic transitions as a surrogate for the real world. In the stochastic "real" environment, there is a small chance at every time step that the agent moves in a random direction instead of the direction it chose. As in the Sect. 6, sim-to-sim experiments allow us to run more experiments than would be possible on a physical robot.

Figure 11 shows GAT and SGAT evaluated for different values of the environment noise parameter. Both the grounding steps and policy improvement steps are repeated until convergence for both algorithms. To evaluate the resulting policy, we estimate the expected return with 10,000 episodes in the "real" environment. At a value of zero, the "real" environment is completely deterministic. At a value of one, every transition is random. Thus, at both of these endpoints, there is no distinction between the expected return gained by the two algorithms.

For every intermediate value, SGAT outperforms GAT. The policy trained using GAT is unaware of the stochastic transitions, so it always takes the shortest and most dangerous path. Meanwhile the SGAT agent learns as if it were training directly on the real environment in the presence of stochasticity. Though Cliff Walking is a relatively simple domain, this experiment demonstrates the importance of modelling the stochasticity in $\mathcal{M}$.

8.2 MuJoCo domains

Having shown the efficacy of SGAT in a tabular domain, we now evaluate its performance in continuous control domains that are closer to real world robotics settings. We perform experiments on the OpenAI Gym MuJoCo environments to compare the effectiveness of SGAT and GAT when there is added noise in the target domain. We consider the case with just

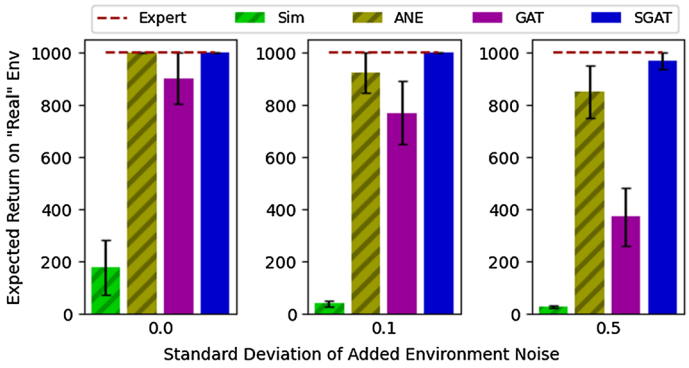


Fig. 12 Sim-to-NoisyReal experiment on **InvertedPendulum**. The "real" pendulum is 10 times heavier than the sim pendulum and has added Gaussian noise of different values. Error bars show standard error over ten independent training runs. Algorithms with striped bars used no real world data during training. SGAT performs comparatively better in noisier target environments

added noise and the case with both noise and domain mismatch between the source and target environments. We call the former Sim-to-NoisySim and the latter Sim-to-NoisyReal. We use the *InvertedPendulum* and *HalfCheetah* domains to test SGAT in environments with both low and high dimensional state and action spaces. For policy improvement, we use an implementation of Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a), from the stable-baselines repository (Hill et al., 2018) with the default hyperparameters for the respective domains.

For GAT, we use a neural network function approximator with two fully connected hidden layers of 64 neurons to represent the forward and inverse models. For SGAT, the forward model outputs the parameters of a Gaussian distribution from which we sample the predicted next state.[7] In our implementation, the final dense layer outputs the mean, μ, and the log standard deviation, $log(\sigma)$, for each element of the state vector. We include all state variables as state variables of interest.

We also compare against the ANE approach from Sect. 6. This baseline is useful in showing that SGAT is accomplishing more than simply adding noise to the actions from the policy. We note the comparison is not a perfectly fair comparison in the sense that robustness approaches such as ANE are sensitive to user-defined hyperparameters that predict the variation in the environment—in this case, the magnitude of the added noise. SGAT automatically learns the right amount of stochasticity from real world data. In these experiments, we chose the ANE hyperparameters (e.g., noise value) with a coarse grid search.

We simulate stochasticity in the target domains by adding Gaussian noise with different standard deviation values to the actions input into the environment. We omit the results of Sim-to-NoisySim experiments for *InvertedPendulum* because both algorithms performed well on the transfer task. Figure 12 shows the performance on the "real" environment of policies trained four ways—naively on the ungrounded simulator, with SGAT, with GAT, and with ANE. In this Sim-to-NoisyReal experiment, SGAT performs much better than GAT when the stochasticity in the target domain increases. Figure 13 shows the same experiment on

[7] A Mixture Density Network might be more suitable when the environment's transition dynamics exhibit multi-modal behavior.

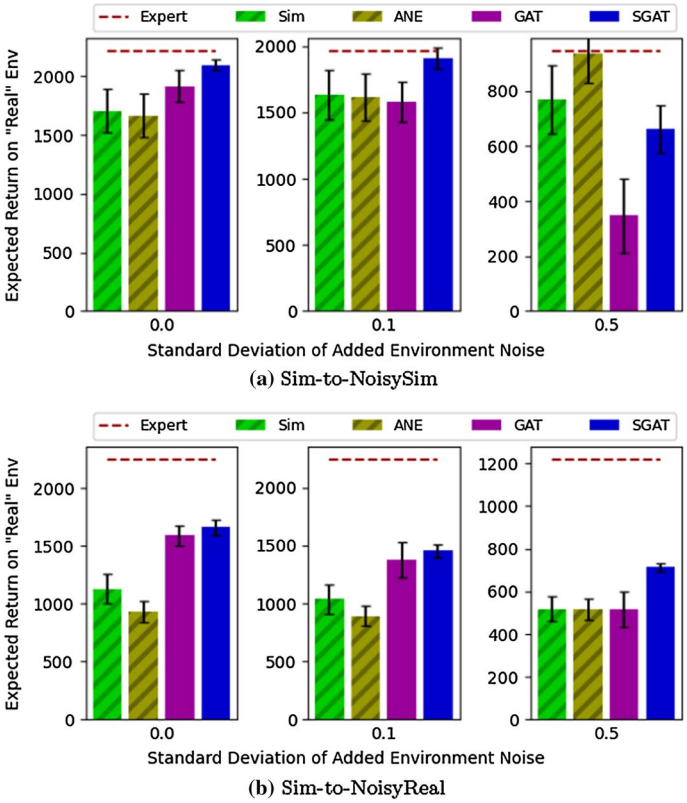


Fig. 13 Sim-to-NoisySim and Sim-to-NoisyReal experiments on **HalfCheetah**. In the NoisyReal environment, the "real" HalfCheetah's mass is 43% greater than the sim HalfCheetah. Error bars show show standard error over ten independent training runs. Algorithms with striped bars used no real world data during training. When the "real" environment is highly stochastic, SGAT performs better than GAT. Meanwhile, ANE does poorly on less noisy scenarios

HalfCheetah, both with and without domain mismatch. Both these environments have an action space of $[-1, 1]$.

The red dashed lines show the performance of a policy trained directly on the "real" environment until convergence, approximately the best possible performance. The axes are scaled respective to this line. The error bars show the standard error across 10 trials with different initialization weights. As the stochasticity increases, SGAT policies perform better than those learned using GAT. Meanwhile, ANE does well only for particular noise values, depending on its training hyperparameters.

8.3 NAO robot experiments

Until this point in our analysis of SGAT, we have used a modified version of the simulator in place of the "real" world so as to isolate the effect of stochasticity (as opposed to domain mismatch). However, the true objective of this research is to enable transfer to real robots, which may exhibit very different noise profiles than the simulated environments. Thus, in this section, we validate SGAT on a real humanoid robot learning to walk on uneven terrain.

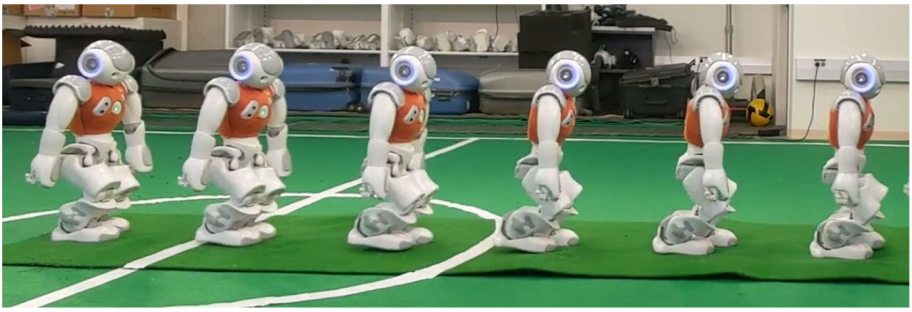


Fig. 14 Experiment setup showing a robot walking on the uneven ground. The NAO begins walking 40 cms behind the center of the circle and walks 300 cms. This image shows a successful walk executed by the robot at 2 sec intervals, learned using the proposed SGAT algorithm

Table 4 Speed and stability of NAO robot walking on uneven ground. The initial policy θ_0 walks at 14.66 ± 1.65 cm/s and always falls down. Both SGAT and GAT find policies that are faster, but SGAT policies are more stable than policies learned using GAT

	Grounding Step 1		Grounding Step 2	
	Speed (cm/s)	Falls	Speed (cm/s)	Falls
GAT	15.7 ± 2.98	6/10	18.5 ± 3.63	10/10
SGAT	16.9 ± 0.678	**0/10**	18.0 ± 2.15	**1/10**

Bold values indicate best performance

As before, we use the NAO robot. We compared GAT and SGAT by independently learning control policies using these algorithms to walk on uneven terrain, as shown in Fig. 14. To create an uneven surface, we placed foam packing material under the turf of a robot soccer field. On this uneven ground, the walking dynamics become more random, since the forces acting on the foot are slightly different every time the robot takes a step. We use the same initial policy as in Sect. 6.5. This initial unoptimized policy achieves a speed of 14.66 ± 1.65 cm/s on the uneven terrain. Aside from these details, the empirical set-up for this task is the same as in Sect. 6.5.

On flat ground, both methods produced very similar policies, but on the uneven ground, the policy learned using SGAT was more successful than a policy learned using GAT. We evaluated the best policy learned using each of SGAT and GAT after each grounding step by generating 10 trajectories on the physical robot. The average speed of the robot on the uneven terrain is shown in Table 4. Qualitatively, we find that the policy learned using SGAT takes shorter steps and stays upright, thereby maintaining its balance on the uneven terrain, whereas the policy produced using GAT learned to lean forward and walk faster, but fell down more often due to the uneven terrain. Both algorithms produce policies that improve the walking speed across grounding steps. The GAT policy after the second grounding step always falls over, whereas the SGAT policy was more stable and finished the course 9 out of 10 times. Overall, this experiment

demonstrates that SGAT allows sim-to-real transfer when the real world is stochastic. Though GAT is able to improve the initial policy's walking speed it is more unstable since it ignores stochasticity in the real world.

9 Discussion of limitations

In this section, we discuss limitations of the GAT and SGAT algorithms and our empirical evaluation. GAT requires that there exists an action that can be taken in simulation to cause the simulator to behave as the real world would. Formally, $\exists \hat{a} \in \mathcal{A}$ such that $f(s,a) = f_{\texttt{sim}}(s,\hat{a})$ for state s and action a. At a minimum this condition should hold for states and actions that are encountered during policy optimization. This requirement is also problematic for domains with P that have high variance in the next state variables and maximum likelihood prediction may be insufficient. However, SGAT provides an alternative algorithm for such cases. Furthermore, both algorithms perform similarly on deterministic environments, suggesting that SGAT should be the default option.

We evaluated GAT on several robot reinforcement learning tasks in both simulation and the real world. In these experiments, we varied the task, policy representation, and the simulator and target MDP (either the real world or another simulator). However, there remain a large number of experimental knobs that we have not studied the importance of yet. Some of these include the reward function definitions, the RL algorithm used, and how the state variables of interest were defined. Further studies of these settings would broaden the breadth of conclusions we can draw about the general applicability of the GAT algorithm.

In this work, we have only considered deterministic simulators, but simulators may have stochastic transitions as well, especially if the simulator was designed to anticipate process noise. However, when using an action transformer grounding approach, stochastic simulators make the learning problem more difficult. We can no longer sample from the distribution provided by the forward model. Instead, the inverse model must take in a distribution over states and output a distribution over actions.

10 Future work

This article introduced an algorithm, grounded action transformation (GAT), that allows a reinforcement learning agent to learn with simulated data. In this section, we propose directions for future research on and application of our new algorithm.

10.1 Sim-to-real in non-robotics domains

We evaluated GAT on a physical NAO robot. GAT is not specific to the NAO and could be applied on other robotics tasks or even non-robotics tasks where a simulator is available a priori. The latter is of particular interest as the sim-to-real problem has been studied to a much lesser extent in non-robotics domains. GAT is most applicable in tasks where the dynamics have a basis in physics and actions have a direct effect on some state variables. For example, if a robot increases the force with which it lifts its arm then it will see its arm lift higher or faster. In such settings, it is reasonable to assume that an effective action grounding function can be learned. It may be less applicable where the dynamics are derived from other factors such as human behavior.

10.2 Identifying state variables of interest

In our empirical evaluation, we manually chose the state variables of interest and modified actions to make the transitions of these variables more realistic. For instance, in Sec. 6.5, we knew it was important to model the effect of actions on the joint positions of the physical robot. Thus, we set the joint positions of the physical robot as the variables of interest. Automatically identifying these variables is an interesting direction for future work.

The state variables of interest should be variables that affect task reward and the goal is to identify them through the data collected for grounding. It may be difficult to identify these variables by simply running an initial policy; more exploratory actions may need to be taken during data collection. Another objective when identifying these variables is that the set of target variables should have minimal size while being large enough for the simulator to be sufficiently grounded for learning to progress. Clearly, setting all state variables to be target variables accomplishes the latter but the grounding problem may become more difficult. Thus methods for automatically identifying the variables should attempt to find the minimal set that still allows learning in the grounded simulation to transfer to the real world.

10.3 Grounded action transformation for deep reinforcement learning

Finally, our empirical evaluation considered relatively low dimensional policy representations: neural networks with a couple of hidden layers, linear functions, or existing parameterized controllers. Some of the most impressive, recent RL success stories have been accomplished with high dimensional neural network policy representations taking pixels as inputs. Applying GAT and SGAT to learn pixel-to-control policies is an interesting and challenging direction for future work. With more complex policy representations there is more chance that the RL algorithm will overfit to the simulator and thus high fidelity grounding is essential. Thus, more complex policy representations and deep RL algorithms are an important test of GAT's ability to ground a simulator.

11 Conclusion

We have introduced an algorithm which allows a robot to learn a policy in a simulated environment and the resulting policy transfer to the physical robot. This algorithm, called the grounded action transformation (GAT) algorithm, makes a contribution towards allowing reinforcement learning agents to leverage simulated data to learn policies that are effective in the real world. We empirically evaluated GAT on three robot learning tasks using the simulated or physical NAO robot. In all cases, GAT leads to higher task performance compared to no grounding. We also compared GAT to a simulator randomization baseline and found that using real world data to modify the simulation was more effective than simply adding noise to the robot's actions during learning. We applied GAT to optimizing the parameters of an existing walk controller and learned the fastest stable walk that we know of for the NAO robot. Finally we also developed a generalization of GAT, SGAT, that improves upon GAT when the real world is highly stochastic.

Acknowledgements We would like to thank Matthew Hausknecht and Patrick MacAlpine for insightful discussions and the anonymous AAAI, IROS, and MLJ reviewers for helpful comments.

Funding This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CPS-1739964, IIS-1724157, NRI-1925082), the Office of Naval Research (N00014-18-2243), Future of Life Institute (RFP2-000), Army Research Office (W911NF-19-2-0333), DARPA, Lockheed Martin, General Motors, and Bosch. The views and conclusions contained in this document are those of the authors alone.

Declaration

Conflict of interest Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Abbeel, P., Quigley, M., & Ng, A. Y. (2006). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on machine learning (ICML)*. http://dl.acm.org/citation.cfm?id=1143845.

Ashar, J., Ashmore, J., Hall, B., Harris, S., Hengst, B., Liu, R., Mei, Z., Pagnucco, M., Roy, R., Sammut, C., Sushkov, O., Teh, B., & Tsekouras, L. (2015). RoboCup SPL 2014 champion team paper. In R. A. C. Bianchi, H. L. Akin, S. Ramamoorthy, K. Sugiura (Eds.) *RoboCup 2014: Robot World Cup XVIII, lecture notes in artificial intelligence* (Vol. 8992, pp. 70–81). Springer International Publishing.

Boeing, A., & Bräunl, T. (2012). Leveraging multiple simulators for crossing the reality gap. In *Proceedings of the 12th international conference on control automation robotics & vision (ICARCV)* (pp. 1113–1119). IEEE.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., & Vanhoucke, V. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., & Fox, D. (2019). Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., & Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. arXiv preprint arXiv:161003518.

Cully, A., Clune, J., Tarapore, D., & Mouret, J. B. (2015). Robots that can adapt like animals. *Nature*, *521*(7553), 503.

Cutler, M., & How, J. P. (2015). Efficient reinforcement learnng for robots using informative simulated priors. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Cutler, M., Walsh, T. J., & How, J. P. (2014). Reinforcement learning with multi-fidelity simulators. In *Proceedings of the IEEE conference on robotics and automation (ICRA)*. http://www.research.rutgers.edu/~thomaswa/pub/icra2014Car.pdf.

Deisenroth, M. P., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th international conference on machine learning (ICML)*.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., & Levine, S. (2017). Learning modular neural network policies for multi-task and multi-robot transfer. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 2169–2176). IEEE.

Fang, K., Bai, Y., Hinterstoisser, S., & Kalakrishnan, M. (2018). Multi-task domain adaptation for deep learning of instance grasping from simulation. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Farchy, A., Barrett, S., MacAlpine, P., & Stone, P. (2013). Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 12th international conference on autonomous agents and multiagent systems (AAMAS)*. http://www.cs.utexas.edu/users/ai-lab/?AAMAS13-Farchy.

Golemo, F., Taiga, A. A., Courville, A., & Oudeyer, P. Y. (2018). Sim-to-real transfer with neural-augmented robot simulation. In *Proceedings of the 2nd conference on robot learning (CORL)* (pp. 817–828).

Hall, B., Harris, S., Hengst, B., Liu, R., Ng, K., Pagnucco, M., Pearson, L., Sammut, C., & Schmidt, P. (2016). RoboCup SPL 2015 champion team paper. In *RoboCup 2015: Robot World Cup XIX, lecture notes in artificial intelligence* (Vol. 9513, pp. 72–82). Springer International Publishing.

Hansen, N., Müller, S. D., & Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, *11*(1), 1–18.

Hengst, B. (2014). *rUNSWift walk2014 report robocup standard platform league*. Technical report. The University of New South Wales.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., & Wu, Y. (2018). Stable baselines. https://github.com/hill-a/stable-baselines.

Iocchi, L., Libera, F. D., & Menegatti, E. (2007). Learning humanoid soccer actions interleaving simulated and real data. In *Proceedings of the 2nd workshop on humanoid soccer robots*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.139.4931.

Jakobi, N., Husbands, P., Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *Proceedings of the European conference on artificial life* (pp. 704–720). Springer.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., & Bousmalis, K. (2019). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE international conference on computer vision and pattern recognition (CVPR)* (pp. 12627–12637).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*. http://ijr.sagepub.com/content/early/2013/08/22/0278364913495721.abstract.

Koos, S., Mouret, J. B., & Doncieux, S. (2010). Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on genetic and evolutionary computation (GECCO)* (pp. 119–126). ACM. http://dl.acm.org/citation.cfm?id=1830505.

Lee, G., Srinivasa, S. S., & Mason, M. T. (2017) GP-ILQG: Data-driven robust optimal control for uncertain nonlinear dynamical systems. arXiv preprint arXiv:170505344.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971.

Lowrey, K., Kolev, S., Dao, J., Rajeswaran, A., & Todorov, E. (2018). Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In P*Proceedings of the IEEE international conference on simulation, modeling, and programming for autonomous robots (SIMPAR)* (pp. 35–42). IEEE.

Marco, A., Berkenkamp, F., Hennig, P., Schoellig, A. P., Krause, A., Schaal, S., & Trimpe, S. (2017). Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 1557–1563). IEEE.

Miglino, O., Lund, H. H., & Nolfi, S. (1996). Evolving mobile robots in simulated and real environments. *Artificial Life*, *2*(4), 417–434.

Molchanov, A., Chen, T., Hönig, W., Preiss, J. A., Ayanian, N., & Sukhatme, G. S. (2019). Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. arXiv preprint arXiv:190304628.

Mozifian, M., Higuera, J.C.G., Meger, D., & Dudek, G. (2019). Learning domain randomization distributions for transfer of locomotion policies. arXiv preprint arXiv:190600410.

Muratore, F., Treede, F., Gienger, & M., Peters, J. (2018). Domain randomization for simulation-based policy optimization with transferability assessment. In *Proceedings of the 2nd conference on robot learning (CORL)* (pp. 700–713).

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., & Zaremba, W. (2018). Learning dexterous in-hand manipulation. arXiv preprint arXiv:180800177.

Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2017). Sim-to-real transfer of robotic control with dynamics randomization. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)*.

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., & Abbeel, P. (2017a). Asymmetric actor critic for image-based robot learning. In *Proceedings of the robotics: Science and systems Conference (RSS)*.

Pinto, L., Davidson, J., Sukthankar, R., Gupta, A. (2017b). Robust adversarial reinforcement learning. In *Proceedings of the 34th international conference on machine learning (ICML)*.

Puterman, M. L. (2014). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons

Rajeswaran, A., Ghotra, S., Levine, S., & Ravindran, B. (2017). EPOpt: Learning robust neural network policies using model ensembles. In *Proceedings of the international conference on learning representations (ICLR)*.

Ramos, F., Possas, R. C., & Fox, D. (2019). Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. arXiv preprint arXiv:190601728.

Rodriguez, D., Brandenburger, A., & Behnke, S. (2019). Combining simulations and real-robot experiments for bayesian optimization of bipedal gait stabilization. In *RoboCup 2018: Robot World Cup XXII, lecture notes in artificial intelligence* (Vol. 11374). Springer International Publishing.

Ross, S., Gordon, G. J., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the international conference on artificial intelligence and statistics (AISTATS)* (pp. 627–635).

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., & Hadsell, R. (2016a). Progressive neural networks. arXiv preprint arXiv:160604671.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., & Hadsell, R. (2016b). Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the 1st conference on robot learning (CORL)*.

Sadeghi, F., & Levine, S. (2017). $(CAD)^2$ RL: Real single-image flight without a single real image. In *Proceedings of the robotics: Science and systems conference (RSS)*.

Schulman, J., Levine, S., Moritz, P., Jordan, M., & Abbeel, P. (2015a). Trust region policy optimization. In *Proceedings of the 32nd international conference on machine learning (ICML)*. http://jmlr.csail.mit.edu/proceedings/papers/v37/schulman15.html.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the international conference on learning representations (ICLR)*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., & Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of the robotics: Science and systems conference (RSS)*.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the 30th IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Tobin, J., Zaremba, W., & Abbeel, P. (2018). Domain randomization and generative models for robotic grasping. In *Proceedings of the 31st IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Tzeng, E., Coline, D., Hoffman, J., Finn, C., Xingchao, P., Levine, S., Saenko, K., & Darrell, T. (2016). Towards adapting deep visuomotor representations from simulated to real environments. In *Proceedings of the workshop on algorithmic foundations of robotics (WAFR)*.

Urieli, D., MacAlpine, P., Kalyanakrishnan, S., Bentor, Y., & Stone, P. (2011). On optimizing interdependent skills: A case study in simulated 3D humanoid robot soccer. In *Proceedings of the 10th international conference on autonomous agents and multiagent systems (AAMAS)* (pp. 769–776).

Zhang, F., Leitner, J., Upcroft, B., & Corke, P. (2016). Vision-based reaching using modular deep networks: From simulation to the real world. arXiv preprint arXiv:161006781.

Zhu, S., Kimmel, A., E Bekris, K., & Boularias, A. (2018). Fast model identification via physics engines for data-efficient policy search. In *Proceedings of the 27th international joint conference on artificial intelligence (IJCAI)* (pp. 3249–3256). https://doi.org/10.24963/ijcai.2018/451.

Authors and Affiliations

Josiah P. Hanna[1] · Siddharth Desai[2] · Haresh Karnan[2] · Garrett Warnell[3] · Peter Stone[4]

[1] School of Informatics, The University of Edinburgh, Edinburgh, UK

[2] Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX, USA

[3] Army Research Laboratory and the Department of Computer Science, The University of Texas at Austin, Austin, TX, USA

[4] Department of Computer Science, The University of Texas at Austin and Sony AI, Austin, TX, USA