Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Grounded Action Transformation for Robot Learning in Simulation

Josiah P. Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{jphanna,pstone}@cs.utexas.edu

Abstract

Robot learning in simulation is a promising alternative to the prohibitive sample cost of learning in the physical world. Unfortunately, policies learned in simulation often perform worse than hand-coded policies when applied on the physical robot. Grounded simulation learning (GSL) promises to address this issue by altering the simulator to better match the real world. This paper proposes a new algorithm for GSL – Grounded Action Transformation – and applies it to learning of humanoid bipedal locomotion. Our approach results in a 43.27% improvement in forward walk velocity compared to a state-of-the art hand-coded walk. We further evaluate our methodology in controlled experiments using a second, higher-fidelity simulator in place of the real world. Our results contribute to a deeper understanding of grounded simulation learning and demonstrate its effectiveness for learning robot control policies.

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. 2015; Lillicrap et al. 2015). Unfortunately, a large gap exists between what is possible in simulation and the reality of robot learning. 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 and How 2015; Koos, Mouret, and Doncieux 2010). Simulation is a valuable tool for robotics research as execution of a robotic skill in simulation is comparatively easier than real world execution. However, the value of simulation learning is limited by the inherent inaccuracy of simulators in modeling the dynamics of the physical world (Kober, Bagnell, and Peters 2013). As a result, learning that takes place in a simulator is unlikely to improve real world performance.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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 paper 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 paper introduces a new algorithm – **G**rounded **A**ction **T**ransformation (GAT) – for simulator grounding within the GSL framework. The algorithm is fully implemented and evaluated using a high-fidelity simulator as a surrogate for the real world. The results of this study contribute to a deeper understanding of transfer from simulation methods and the effectiveness of GAT. In contrast to prior work, our real-world evaluation of GAT starts from a state-of-the-art walk engine as the base policy, and nonetheless is able to improve the walk velocity by over 43%, leading to what may be the fastest known walk on the SoftBank NAO robot.

Background

Preliminaries

We formalize robot skill learning as a *reinforcement learning* (RL) problem (Sutton and Barto 1998). At discrete time-step t the robot takes action $A_t \sim \pi(\cdot|S_t)$ according to a policy, π, which is a distribution over actions, $A_t \in \mathcal{A}$, conditioned on the current state, $S_t \in \mathcal{S}$. The environment, E, responds with $S_{t+1} \sim P(\cdot|S_t, A_t)$ according to the dynamics, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$ which is a probability density function over next states conditioned on the current state and action. A trajectory of length L is a sequence of states and actions, $\tau := S_0, A_0, ..., S_L, A_L$. We also define a cost function, c, which assigns a scalar cost to each (s, a) pair. We denote the value of policy, π, as $J(\pi) := \mathbb{E}_{\tau \sim Pr(\cdot|\pi)} \left[\sum_{t=0}^{L} c(S_t, A_t) \right]$ where $Pr(\tau|\pi)$ is the probability of τ when selecting actions according to π.

We assume π is parameterized by a vector $\boldsymbol{\theta}$ and denote the parameterized policy as $\pi_{\boldsymbol{\theta}}$. Since $\boldsymbol{\theta}$ determines the policy distribution we overload notation by referring to $\pi_{\boldsymbol{\theta}}$ as $\boldsymbol{\theta}$. Given initial parameters $\boldsymbol{\theta}_0$, the goal of policy improvement is to find $\boldsymbol{\theta}'$ such that $J(\boldsymbol{\theta}') < J(\boldsymbol{\theta}_0)$. In this work, $\boldsymbol{\theta}$ is a deterministic policy that maps state observations to an action

vector $\mathbf{a}_t$. Each component of $\mathbf{a}_t$, $\mathbf{a}_t^i$, is the desired joint angle for degree-of-freedom i. At each time-step, t, $\boldsymbol{\theta}$ selects $\mathbf{a}_t$ according to the robot's configuration in joint space, $\mathbf{x}_t$, high level intention commands (e.g., walk forward at 75% of maximum velocity), $\boldsymbol{\omega}_t$, and a sensor vector, $\boldsymbol{\psi}_t$, of inertial measurement (IMU) and foot sensor readings. The state of the robot can be fully described as $\mathbf{s}_t = \langle \mathbf{x}_t, \dot{\mathbf{x}}_t, \ddot{\mathbf{x}}_t, \boldsymbol{\omega}_t, \boldsymbol{\psi}_t \rangle$ where $\dot{\mathbf{x}}_t$ and $\ddot{\mathbf{x}}_t$ are the time derivatives of $\mathbf{x}_t$. Since the robot only observes $\mathbf{x}_t$, $\boldsymbol{\omega}_t$, and $\boldsymbol{\psi}_t$, E is partially observable. Note that while the high level intention commands are determined by the robot, from the point-of-view of $\boldsymbol{\theta}$, these directional commands are state features.

In this paper, learning takes place in a simulator which is an environment, $E_{\texttt{sim}}$, that models E. Specifically $E_{\texttt{sim}}$ has the same state-action space as E but inevitably a different dynamics distribution, $P_{\texttt{sim}}$. In many robotics settings, c is user defined and thus is identical for E and $E_{\texttt{sim}}$. However, the different dynamics distribution mean that for any policy, $\boldsymbol{\theta}$, $J(\boldsymbol{\theta}) \neq J_{\texttt{sim}}(\boldsymbol{\theta})$ since $\boldsymbol{\theta}$ induces a different trajectory distribution in E than in $E_{\texttt{sim}}$. Thus $\boldsymbol{\theta}'$ with $J_{\texttt{sim}}(\boldsymbol{\theta}') < J_{\texttt{sim}}(\boldsymbol{\theta}_0)$ does not imply $J(\boldsymbol{\theta}') < J(\boldsymbol{\theta}_0)$ – in fact $J(\boldsymbol{\theta}')$ could even be worse than $J(\boldsymbol{\theta}_0)$. In practice and in the literature, learning in a simulator frequently leads to catastrophic degradation of J. This paper explores methods for learning in $E_{\texttt{sim}}$ that result in lower policy cost.

Related Work

This section surveys previous research in simulation-transfer methods. Our work also relates to model-based reinforcement learning, however, we restrict our attention here to methods directly concerned with learning in simulation.

One approach to simulation-transfer is using experience in simulation to reduce learning time on the physical robot. 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. 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 et al. (2015) use experience in simulation to estimate a prior for a Gaussian process model to be used with the PILCO learning algorithm (Deisenroth and Rasmussen 2011). Rusu et al. (2016a; 2016b) introduce progressive neural network policies which are initially trained in simulation before a final period of learning in the true environment. In contrast to these methods, our method performs all learning in a grounded simulator and only uses the physical robot for policy evaluation.

Another class of simulation-transfer methods optimize an objective besides $J_{\texttt{sim}}$ in simulation. Rajeswaran et al. (2016) use a set of different simulators to learn robust policies that can perform well in a variety of environments. Koos et al. (2010) use multi-objective optimization to find policies that trade off between optimizing $J_{\texttt{sim}}(\pi)$ and *transferability*. Iocchi et al. (2007) compute a correction function from $J(\pi)$ such that $J_{\texttt{sim}}(\pi) = J(\pi)$ and then optimize the corrected objective. In contrast, we directly optimize $J_{\texttt{sim}}$ within a modified simulator.

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. This approach is complementary to ours although it also requires a simulator which can be reset to an arbitrary state during policy execution.

Grounded Simulation Learning

In this section we introduce the Grounded Simulation Learning (GSL) framework as presented by Farchy et al. (2013). GSL improves robot learning in simulation by using state transition data from the physical system to modify $E_{\texttt{sim}}$ such that the modified $E_{\texttt{sim}}$ is a higher fidelity model of E. 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, $E_{\texttt{sim}} = \langle \mathcal{S}, \mathcal{A}, P_{\texttt{sim}}, c \rangle$, that models the environment of interest, E. Furthermore, $E_{\texttt{sim}}$ must be *modifiable*. A modifiable 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. Additionally, GSL assumes the physical robot can record a data set, $\mathcal{D}$ of trajectories when executing any policy $\boldsymbol{\theta}$.
3. Finally, GSL requires a policy improvement routine, `optimize`, that searches for $\boldsymbol{\theta}$ which decreases $J_{\texttt{sim}}(\boldsymbol{\theta})$. The `optimize` routine returns a set of candidate policies, Π_c to evaluate on the physical robot.

Let $d(p,q)$ be a measure of similarity between probabilities p and q. GSL grounds $E_{\texttt{sim}}$ by finding $\boldsymbol{\phi}^\star$ such that:

$$\boldsymbol{\phi}^\star = \operatorname*{argmin}_{\boldsymbol{\phi}} \sum_{\tau \in \mathcal{D}} d\left(Pr(\tau|\boldsymbol{\theta}), Pr_{\texttt{sim}}(\tau|\boldsymbol{\theta}, \boldsymbol{\phi})\right) \quad (1)$$

where $Pr(\tau|\boldsymbol{\theta})$ is the probability of observing trajectory τ on the physical robot and $Pr_{\texttt{sim}}(\tau|\boldsymbol{\theta}, \boldsymbol{\phi})$ is the probability of τ in the simulator with dynamics parameterized by $\boldsymbol{\phi}$. For instance, if $d(p,q) := \log p - \log q$ for two probabilities p and q then $\boldsymbol{\phi}^\star$ minimizes the Kullback-Leibler divergence between $Pr(\cdot|\boldsymbol{\theta})$ and $Pr_{\texttt{sim}}(\cdot|\boldsymbol{\theta}, \boldsymbol{\phi}^\star)$.

Assuming GSL can solve (1), the framework is as follows:

1. Execute policy $\boldsymbol{\theta}_0$ on the physical robot to collect a data set of trajectories, $\mathcal{D}$.
2. Use $\mathcal{D}$ to find $\boldsymbol{\phi}^\star$ that satisfies Equation 1.
3. Use `optimize` with $J_{\texttt{sim}}$ and $P_{\boldsymbol{\phi}^\star}$ to learn a set of candidate policies Π_c in simulation which are expected to perform well on the physical robot.
4. Evaluate each proposed $\boldsymbol{\theta}_c \in \Pi_c$ on the physical robot and return the policy, $\boldsymbol{\theta}_1$, with minimal J.

GSL can be applied iteratively with $\boldsymbol{\theta}_1$ being used to collect more trajectories to ground the simulator again before learning $\boldsymbol{\theta}_2$. The re-grounding step is necessary since changes to $\boldsymbol{\theta}$ result in changes to the distribution of states that the robot observes. When this happens, a simulator that has been modified with data from the state distribution of $\boldsymbol{\theta}_0$ may be a poor model under the state distribution of $\boldsymbol{\theta}_1$.

Farchy et al. proposed a GSL algorithm, which we refer to as GUIDED GSL. GUIDED GSL blends simulator modification with human guided policy optimization. This algorithm demonstrated the efficacy of GSL in optimizing forward walk velocity of a bipedal robot. The robot in that research began learning with a slow but stable walk policy. Four iterations of GUIDED GSL led to an improvement of over 26% on this baseline walk. Two limitations of GUIDED GSL are that:

1. It makes the assumption that actions in simulation achieve their desired effect instantaneously.
2. It required a human expert to manually select which components of $\boldsymbol{\theta}$ were allowed to change between iterations of the `optimize` routine.

In this work, we introduce an enhanced GSL methodology that eliminates both assumptions and optimizes one of the fastest available NAO walk engines.

Robot Platform and Task

Before presenting GAT, our new GSL algorithm, we describe the robot platform, initial walk policy, and simulators which we use for evaluation. While our method is applicable to other robots, tasks, and simulators we describe these components first in order to ground the algorithm's presentation.

Robot Platform: Our target task is bipedal walking using the SoftBank NAO.[1] The NAO is a humanoid robot with 25 degrees of freedom (See Figure 2a). For walking, our NAO uses an open source walk engine developed at the University of New South Wales (UNSW) (Ashar et al. 2015; Hall et al. 2016). This walk engine has been used by at least one team in each of the past three 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 one of the fastest open source walks available.[2] The walk is a zero moment point walk based on an inverted pendulum model. The walk is closed loop, using the inertial measurement (IMU) sensors and a learned ankle control policy to improve stability. The UNSW walk engine has 15 parameters that determine features of the walk (e.g., step height, pendulum model height). The values of the parameters from the open source release constitute $\boldsymbol{\theta}_0$. For full information on the UNSW walk see (Hengst 2014).

Simulators: We use two different simulators in this work. The first is the open source SimSpark simulator.[3] The simulator simulates realistic physics with the Open Dynamics Engine.[4] This simulator was also used in the work introducing GSL (Farchy et al. 2013). We also use the Gazebo simulator[5] provided by the Open Source Robotics Foundation.[6] Gazebo is an open source simulator that is commonly used in robotics research. SimSpark enables fast simulation but is a lower fidelity model of the real world. Gazebo enables high fidelity simulation with an additional computational cost. In one of our experiments we consider the more realistic Gazebo environment as E and SimSpark as $E_{\texttt{sim}}$. The main difference between these simulators and the physical robot is how actions change the robot's configuration. In SimSpark, actions achieve the desired command angle almost instantaneously. On the physical robot there is a delay. Gazebo also models joints as more reactive than the real world although less reactive than in SimSpark.

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

[2] While other regular RoboCup participants also have competitive walks, we are not aware of any published result that confirms a speed as much as 42% faster than UNSW's, as we achieve in this paper.

[3] http://simspark.sourceforge.net

[4] http://www.ode.org/

[5] http://gazebosim.org/

[6] http://www.osrfoundation.org/

Grounded Action Transformation

We now introduce our principle algorithmic contribution, GSL with *Grounded Action Transformation* (GAT) which improves the grounding step (step 2) of the GSL framework. GAT improves grounding by correcting differences in the effects of actions between E and $E_{\texttt{sim}}$. GSL with GAT is presented in Algorithm 1.

The GSL framework grounds the simulator by finding $\boldsymbol{\phi}^\star$ that satisfies (1). Since it is often easier to minimize error in the one step dynamics distribution than error in the trajectory distributions, GAT uses:

$$\boldsymbol{\phi}^\star = \operatorname*{argmin}_{\boldsymbol{\phi}} \sum_{\tau_i \in \mathcal{D}} \sum_{t=0}^{L} d(P(\mathbf{s}^i_{t+1}|\mathbf{s}^i_t, \mathbf{a}^i_t), P_{\boldsymbol{\phi}}(\mathbf{s}^i_{t+1}|\mathbf{s}^i_t, \mathbf{a}^i_t)) \tag{2}$$

as a surrogate loss function which can be minimized with transitions observed in $\mathcal{D}$. GSL with GAT begins by collecting the dataset $\mathcal{D}$ (Algorithm 1, line 4).

Physics-based simulators (such as SimSpark and Gazebo) have a large number of parameters determining the physics of the simulated environment (e.g., friction coefficients). However, using these parameters as $\boldsymbol{\phi}$ is not amenable to numerical optimization of (2). To find $\boldsymbol{\phi}^\star$ efficiently, GAT uses a parameterized *action transformation* function which takes the agent's state and action as input and outputs a new action which – when taken in simulation – will result in the robot transitioning to the same next state it would have in E. We denote this function, $g_{\boldsymbol{\phi}} : \mathcal{S} \times \mathcal{A} \to \mathcal{A}$; the parameters of g serve as $\boldsymbol{\phi}$ under the GSL framework. GAT constructs g with parameterized models of the robot's dynamics and the simulator's *inverse* dynamics. Assuming the simulated robot can record a dataset $\mathcal{D}_{\texttt{sim}}$ of experience like the physical robot, GAT reduces (2) to a supervised learning problem.

GAT defines $g \coloneqq g_{\boldsymbol{\phi}}$ by a deterministic forward model of the robot's dynamics, f and a deterministic model of the simulator's inverse dynamics, $f^{-1}_{\texttt{sim}}$. The function, f maps $(\mathbf{s}_t, \mathbf{a}_t)$ to the maximum likelihood estimate of $\mathbf{x}_{t+1}$ under P. The function $f^{-1}_{\texttt{sim}}$ maps $(\mathbf{s}_t, \mathbf{s}_{t+1})$ to the action that is most likely to produce this transition in simulation. When executing $\boldsymbol{\theta}$ in simulation, the robot selects $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)$ and then uses f to predict what the resulting configuration, $\mathbf{x}_{t+1}$, would be in E. Then $\mathbf{a}_t$ is replaced with $\hat{\mathbf{a}}_t \coloneqq f^{-1}_{\texttt{sim}}(\mathbf{s}_t, f(\mathbf{s}_t, \mathbf{a}_t))$. The result is that the robot achieves the exact $\mathbf{x}_{t+1}$ it would have on the physical robot.[7]

[7] GAT subsumes GUIDED GSL which makes the additional as-

Algorithm 1 Grounded Action Transformation (GAT) Pseudo code. Input: An initial policy, $\boldsymbol{\theta}$, the environment, E, a simulator, $E_{\texttt{sim}}$, smoothing parameter α, and a policy improvement method, `optimize`. The function `rolloutN`$(\boldsymbol{\theta}, \texttt{N})$ executes N trajectories with $\boldsymbol{\theta}$ and returns the observed state transition data. The functions `trainForwardModel` and `trainInverseModel` estimate models of the forward and inverse dynamics respectively.

1: **function** GAT
2: $\quad \boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$
3: $\quad \mathcal{D}_{\texttt{robot}} \leftarrow \texttt{RolloutN}(E, \boldsymbol{\theta}_0, N)$
4: $\quad \mathcal{D}_{\texttt{sim}} \leftarrow \texttt{RolloutN}(E_{\texttt{sim}}, \boldsymbol{\theta}_0, N)$
5: $\quad f \leftarrow \texttt{trainForwardModel}(\mathcal{D}_{\texttt{robot}})$
6: $\quad f^{-1}_{\texttt{sim}} \leftarrow \texttt{trainInverseModel}(\mathcal{D}_{\texttt{sim}})$
7: $\quad g(s,a) \leftarrow \alpha f^{-1}_{\texttt{sim}}(s, f(s,a)) + (1-\alpha) \cdot a_t$
8: $\quad \Pi \leftarrow \texttt{optimize}(E_{\texttt{sim}}, \boldsymbol{\theta}, g)$
9: $\quad$ **return** $\text{argmin}_{\boldsymbol{\theta} \in \Pi} J(\boldsymbol{\theta})$
10: **end function**

In practice f and $f^{-1}_{\texttt{sim}}$ are represented with supervised regression models and learned from $\mathcal{D}$ and $\mathcal{D}_{\texttt{sim}}$ respectively (Algorithm 1 lines 5-6). The approximation of g introduces noise into the robot's motion. To ensure stable motions, GAT uses a smoothing parameter α. The action transformation function (Algorithm 1 line 8) is then defined as:

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

.

In our experiments, we set α as high as possible subject to the walk remaining stable. Figure 1 illustrates the GAT-modified $E_{\texttt{sim}}$. GAT then proceeds to improve $\boldsymbol{\theta}$ with `optimize` within the grounded simulator (lines 8-9).

GAT Implementation

In this work, GAT uses two neural networks to approximate f and $f^{-1}_{\texttt{sim}}$. Each function is a three layer network with 200 hidden units in the first layer and 180 hidden units in the second. During simulator modification, the f network receives $\mathbf{s}_t$ and $\mathbf{a}_t$ as input and the $f^{-1}_{\texttt{sim}}$ network receives $\mathbf{s}_t$ and the output of f as input. The final output from $f^{-1}_{\texttt{sim}}$ is the replacement action $\hat{\mathbf{a}}_t$. While $\mathbf{a}_t$ is a vector of desired joint angles, the action input to f and action output of $f^{-1}_{\texttt{sim}}$ is encoded as a desired change in $\mathbf{x}_t$ which was found to improve prediction. We follow Fu et al. (2015) by having f predict the joint acceleration caused by executing $\mathbf{a}_t$ in $\mathbf{s}_t$ instead of directly predicting $\mathbf{s}_{t+1}$. The accelerations can then be integrated and added to $\mathbf{s}_{t+1}$ to produce the resulting next state. Additionally, f and $f^{-1}_{\texttt{sim}}$ regress to the sine and cosine of the target angular accelerations and then pass the outputs through the $\arctan$ function to produce the final angular acceleration. Passing the network outputs through the $\arctan$ function ensures f and $f^{-1}_{\texttt{sim}}$ produce valid joint angles. The true state, s_t, is estimated by concatenating joint configurations $\mathbf{x}_t, \mathbf{x}_{t-1}, ..\mathbf{x}_{t-4}$ and past actions $\mathbf{a}_{t-1}, ..., \mathbf{a}_{t-4}$. The state estimate $\langle \mathbf{x}_t, ..., \mathbf{x}_{t-4}, \mathbf{a}_{t-1}, ..., \mathbf{a}_{t-4} \rangle$ improves the predictions of f and $f^{-1}_{\texttt{sim}}$ because it implicitly captures the unobserved $\dot{\mathbf{x}}_t$ and $\ddot{\mathbf{x}}_t$ state variables. The length of the configuration history was chosen to balance predictive accuracy with keeping the number of inputs to the networks small. Both networks are trained with backpropagation.

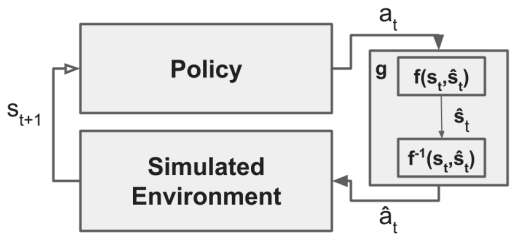


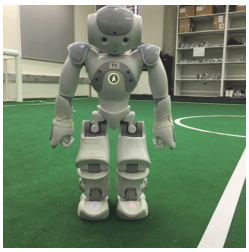
Figure 1: An illustration of the modifiable simulator GAT induces. At each time-step the robot takes an action, $\mathbf{a}_t$, and passes $\mathbf{a}_t$ to a modification function, g. The modification function uses a deterministic model of the real robot's dynamics, f, to predict the effect of executing $\mathbf{a}_t$ on the physical robot. Then, a model of the simulated robot's inverse dynamics uses the prediction, $\hat{\mathbf{x}}_t$, to predict the action $\hat{\mathbf{a}}_t$ which will achieve $\hat{\mathbf{x}}_t$ in simulation. Finally, $\hat{\mathbf{a}}_t$ is passed to the environment, $E_{\texttt{sim}}$ and the resulting state transition will be similar to the transition that would have occurred in E.

Empirical Results

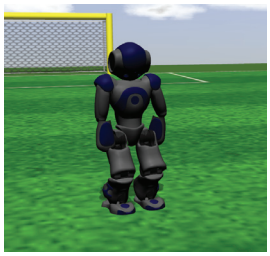
Experimental Set-up

We evaluate GAT on the task of bipedal robot walking. The walk takes a target forward velocity parameter in the range $[0, 1]$. We set this parameter to 0.75 which we found led to the fastest walk that was reliably stable. The robot walks forward towards a target at this velocity. If the robot's angle to the target becomes greater than five degrees it turns back towards the target while continuing to walk forward. In all environments, $J(\boldsymbol{\theta})$ is the negative of the average forward walk velocity while executing $\boldsymbol{\theta}$. On the physical robot a trajectory terminates once the robot has walked four meters or falls. A trajectory generated with $\boldsymbol{\theta}_0$ lasts ≈ 20.5 seconds on the robot. 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. Previous work has shown that when using a model estimated from data it is better to use shorter action trajectories to avoid overfitting to an inaccurate model (Jiang et al. 2015). Even for identical s_t and a_t, s_{t+1} in $E_{\texttt{sim}}$ will most likely be different than s_{t+1} in E. This error compounds over the course of a trajectory since state error at s_t is propagated forward into s_{t+1}. This observation motivates using shorter trajectory lengths as simulator fidelity decreases.

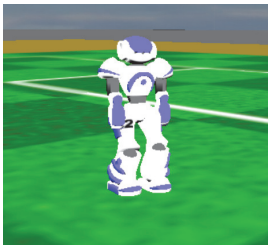
sumption $f^{-1}_{\texttt{sim}}(\mathbf{s}_t, \mathbf{x}_{t+1}) = \mathbf{x}_{t+1}$, in other words that requesting joint angles means the robot achieves those joint angles at the next time-step. GAT removes all human guidance from policy optimization. GAT also removes the need for human expertise in selecting the components of $\boldsymbol{\theta}$ which are allowed to change during policy optimization.



(a) A Softbank NAO Robot



(b) A simulated NAO in Gazebo



(c) A simulated NAO in SimSpark

Figure 2: The three robotic environments used in this work. The Softbank NAO is our target physical robot. The NAO is simulated in the Gazebo and SimSpark simulators.

Optimizing $\boldsymbol{\theta}$ We use the Covariance Matrix Adaption-Evolutionary Strategy (CMA-ES) algorithm (Hansen 2006) as the `optimize` routine. CMA-ES is a policy search method which samples a population of candidate $\boldsymbol{\theta}$ from a Gaussian distribution over parameters. The top k parameter vectors are used to update the sampling distribution so that the mean is closer to an optimal policy. We modify $J_{\texttt{sim}}(\boldsymbol{\theta})$ for the optimization by adding a cost of 15 for any trajectory in which the robot falls. The added penalty encourages CMA-ES to strongly favor stable policies over faster, less stable ones. To clarify terminology, a generation refers to a single update of CMA-ES; an iteration refers to a complete cycle of GAT.

SimSpark to Gazebo: Since a large number of trials are difficult to obtain on a physical robot, we present a study of GAT using Gazebo as a surrogate for the real world. In this setting we evaluate the effectiveness of GAT compared to learning with no grounding and grounding $E_{\texttt{sim}}$ by injecting noise into 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, Husbands, and Harvey 1995). We refer to this baseline as Noise-Envelope. Since GAT with function approximation introduces noise into the robot's actions we wish to verify that GAT offers benefits over such methods. Noise-Envelope adds standard Gaussian noise to the robot's actions within the ungrounded simulator. We also attempted to evaluate GUIDED GSL but preliminary experiments showed that the assumption that actions achieve their desired effect instantaneously did not hold for this setting.

We run 10 trials of each method. 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 optimize $\boldsymbol{\theta}$ for 10 generations of the CMA-ES algorithm. In each generation, 150 policies are sampled and evaluated in simulation. CMA-ES estimates $J_{\texttt{sim}}$ with 20 trajectories from each policy. Overall, the CMA-ES optimization requires 30,000 simulated trajectories for each trial.

Table 1 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. This table only considers policies found after the first generation of CMA-ES. The reason for this is that the policies in the first generation were randomly sampled from an initial search distribution. We consider learning to begin once CMA-ES has used the $J_{\texttt{sim}}$ values of the first generation to update the search distribution. Using $J_{\texttt{sim}}$ to identify $\boldsymbol{\theta}$ which improve J in the first generation is a *policy evaluation* problem while improvement afterwards is a *policy improvement* problem.

Simulation to NAO Set-up: We evaluate GAT for transferring policies learned in Simspark or Gazebo to the physical NAO. The data set $\mathcal{D}$ consists of 15 trajectories collected with $\boldsymbol{\theta}_0$ on the physical NAO. For each iteration, we optimize $\boldsymbol{\theta}$ for 10 generations of the CMA-ES algorithm. We evaluate the best policy from each generation with 5 trajectories on the physical robot. If the robot falls in any of the 5 trajectories the policy is considered unstable. While the number of evaluations is small, in practice stable policies had small variance in walk velocity and unstable policies fell in almost all trajectories.

One challenge in this setting is the simulators receive robot actions at 50 Hz while the physical NAO receives robot actions at 100 Hz. The discrepancy in action frequency poses a problem for using real world data to modify how joints move in simulation. By skipping every other measurement to get an effectively 50Hz data trace, we are able to model how the physical robot's joints move at the simulator's frequency.

Experimental Results

SimSpark to Gazebo Results: Table 1 shows that GAT maximizes improvement in J while minimizing iterations without improvement. NOISE-ENVELOPE 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 NOISE-ENVELOPE. This result demonstrates that action transformations are grounding the simulator in a more effective way than simply injecting noise.

Table 1 also shows that on average GAT finds an improved policy within the first few policy updates after grounding. When learning with no grounding finds an improvement it is also usually in an early generation of CMA-ES. As policy improvement progresses, the best policies in each generation begin to overfit to the dynamics of $E_{\texttt{sim}}$. Without grounding overfitting happens almost immediately. NOISE-ENVELOPE is shown to be more robust to overfitting since any policy it

Method	% Improve	Failures	Best Gen.
No Ground	11.094	7	1.33
Noise-Envelope	18.93	5	6.6
GAT	**22.48**	**1**	2.67

Table 1: This table compares Grounded Action Transformation (GAT) with baseline approaches for transferring learning between SimSpark and Gazebo. 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.

proposes achieved good cost in a noisy $E_{\texttt{sim}}$. The grounding done by GAT is inherently local to the trajectory distribution of $\boldsymbol{\theta}_0$. Thus as $\boldsymbol{\theta}$ changes in policy improvement, the action transformation function fails to produce a more realistic simulator. Noise modification methods can mitigate overfitting by emphasizing robust policies although it is also limited in finding as strong of an improvement as GAT.

Simulator to Physical NAO Results: Our final empirical evaluation applies GAT to learning bipedal walks on a physical NAO. Table 2 gives the walk velocity of policies learned in simulation with GAT. The physical robot walks at a velocity of 19.52 cm/s with $\boldsymbol{\theta}_0$. Two iterations of GAT with SimSpark increased the walk velocity of the NAO to 27.97 cm/s — an improvement of 43.27% compared to $\boldsymbol{\theta}_0$.[8] GAT with SimSpark and GAT with Gazebo both improved walk velocity by over 30%. This result demonstrates generality of our approach across different simulators.

As in the previous experiment, policy improvement with CMA-ES required 30,000 trajectories per 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 $\boldsymbol{\theta}_c \in \Pi_c$). 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. demonstrated the benefits of re-grounding and further optimizing $\boldsymbol{\theta}$. We reground using the 15 trajectories collected with the best policy found by GAT with SimSpark and optimize for a further 10 generations in simulation. The second iteration results in a walk, $\boldsymbol{\theta}_2$, which averages 27.97 cm/s for a total improvement of 43.27% over $\boldsymbol{\theta}_0$.

Finally, we evaluate $\boldsymbol{\theta}_0$ with 100% of maximum velocity requested (i.e., the forward velocity request parameter set to 1.0). The average velocity of this walk across five stable trajectories is 24.3 cm / s. This result shows that GAT can learn walk policies that outperform one of the best hand coded walks available.

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

Method	Velocity (cm/s)	% Improve
$\boldsymbol{\theta}_0$	19.52	0.0
GAT SimSpark $\boldsymbol{\theta}_1$	26.27	34.58
GAT SimSpark $\boldsymbol{\theta}_2$	27.97	43.27
GAT Gazebo $\boldsymbol{\theta}_1$	26.89	37.76

Table 2: This table gives the maximum learned velocity and percent improvement for each method starting from $\boldsymbol{\theta}_0$ (top row).

Discussion and Future Work

Our proposed algorithm, GAT, has some limitations that we discuss here. The decision to learn an action modification function, g, makes the assumption that $\forall(s_t, a_t, s_{t+1})$ that could be observed on the physical robot there exists an action $\hat{a}_t$ that will produce the same transition when used in place of a_t. We posit that this assumption is often true since $\hat{a}_t$ can usually be executed with more or less force in simulation to achieve the desired response. However, this assumption is likely to break down under contact dynamics where external forces resist the robot's actions. Other tasks may introduce other forms of simulator bias that GAT is currently limited in handling. An important direction for future work is to characterize the settings where this approach is limited and to identify alternatives.

We specify simulator grounding as a supervised learning problem however the distribution of inputs to the learned models, f and $f^{-1}_{\texttt{sim}}$, during policy execution differ from the training distributions. The distribution change is likely to weaken the quality of action modification under g. To see why the change happens consider that f is trained under (s_t, a_t, s_{t+1}) sampled from executing $\boldsymbol{\theta}$ on the real robot while $f^{-1}_{\texttt{sim}}$ is trained on (s_t, a_t, s_{t+1}) sampled in simulation. During simulator modification, both models are evaluated on data sampled from the distribution of $\boldsymbol{\theta}$ in the grounded simulator. We have started to explore methods similar to the Dataset Aggregation (DAgger) algorithm (Ross and Bagnell 2012) which collects new data to retrain $f^{-1}_{\texttt{sim}}$ as the state distribution of $\boldsymbol{\theta}$ in $E_{\texttt{sim}}$ changes with grounding.

Finally, this paper considers action modification but a complementary approach could consider sensor modification. Sensor modification would add another layer to the modified simulator such that the environment returns a state and the sensor modification function predicts what the robot would observe in that state. The two methods have an interesting interaction since sensor modification changes the action $\boldsymbol{\theta}$ selects while action modification changes the next state which changes the observed sensor values as well.

Conclusion

This paper proposed and evaluated the Grounded Action Transformation (GAT) algorithm for grounded simulation learning. Our method led to a 43.27 % improvement in the walk velocity of a state-of-the-art bipedal robot. We conducted an empirical study that analyzed the benefits of GAT compared to a pair of baseline simulation-transfer methods. This experiment demonstrates GAT enhances learning in simulation in comparison to other methods.

Acknowledgments

We would like to thank Matthew Hausknecht, Patrick MacAlpine, and Garrett Warnell for insightful discussions and the anonymous reviewers for helpful comments. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), and AFOSR (FA9550-14-1-0087). Josiah Hanna is supported by an NSF Graduate Research Fellowship. Peter Stone serves on the Board of Directors of, Cogitai, Inc. 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

Ashar, J.; Ashmore, J.; Hall, B.; and Harris, S. e. a. 2015. Robocup spl 2014 champion team paper. In *RoboCup 2014: Robot World Cup XVIII*, volume 8992 of *Lecture Notes in Computer Science*. Springer International Publishing. 70–81.

Christiano, P.; Shah, Z.; Mordatch, I.; Schneider, J.; Blackwell, T.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2016. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*.

Cully, A.; Clune, J.; Tarapore, D.; and Mouret, J.-B. 2015. Robots that can adapt like animals. *Nature*.

Cutler, M., and How, J. P. 2015. Efficient reinforcement learnng for robots using informative simulated priors. In *IEEE International Conference on Robotics and Automation, ICRA*.

Cutler, M.; Walsh, T. J.; and How, J. P. 2014. Reinforcement learning with multi-fidelity simulators. In *IEEE Conference on Robotics and Automation, ICRA*.

Deisenroth, M. P., and Rasmussen, C. E. 2011. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning, ICML*.

Farchy, A.; Barrett, S.; MacAlpine, P.; and Stone, P. 2013. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Twelth International Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

Fu, J.; Levine, S.; and Abbeel, P. 2015. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Hall, B.; Harris, S.; Hengst, B.; Liu, R.; Ng, K.; Pagnucco, M.; Pearson, L.; Sammut, C.; and Schmidt, P. 2016. Robocup spl 2015 champion team paper.

Hansen, N. 2006. The CMA evolution strategy: a comparing review. In Lozano, J.; Larranaga, P.; Inza, I.; and Bengoetxea, E., eds., *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Springer. 75–102.

Hengst, B. 2014. runswift walk2014 report robocup standard platform league. Technical report, The University of New South Wales.

Iocchi, L.; Libera, F. D.; and Menegatti, E. 2007. Learning humanoid soccer actions interleaving simulated and real data. In *Second Workshop on Humanoid Soccer Robots*.

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

Jiang, N.; Kulesza, A.; Singh, S.; and Lewis, R. 2015. The dependence of effective planning horizon on model accuracy. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*.

Koos, S.; Mouret, J.-B.; and 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*, 119–126. ACM.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Rajeswaran, A.; Ghotra, S.; Levine, S.; and Ravindran, B. 2016. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*.

Ross, S., and Bagnell, J. A. 2012. Agnostic system identification for model-based reinforcement learning. In *29th International Conference on Machine Learning, ICML*.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016a. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

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

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. I.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.