Holistic Action Transform

Harsh Goyal
Supervisors: Josiah Hanna, Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
hgoyal@cs.utexas.edu

May 23, 2019

1 Abstract

Robots can be expensive and fragile. It can also take them a long time to execute instructions in the physical world. These are some reasons why a simulation environment is often used as a proxy for the real world when having a robot learn how to perform a task well. However, this carries the risk of the robot learning a skill which overfits to the simulator. This overfitting leads to the learnt skill not performing well in the real world. That is why there is previous work which makes efforts to automatically modify the simulator to minimize differences between the simulator and the real environment.

Intuitively, a simulator models an environment well if, for each skill under consideration, there is a correlation between the performances of the physical agent and the simulation agent on the given task. In this work, we introduce Holistic Action Transform (HAT), which lets us encode this intuition about what it means to be a good simulator into the measures of differences between reality and simulation that we minimize with simulator optimization. We then evaluate HAT in a simple “Linear Dynamical System” domain. We establish that minimizing these measures of differences in this domain can increase the correlation between performances of agents in the two environments, which enables the simulation agent to learn skills that transfer well to the real world. Finally, we make empirical comparisons between HAT and previous work. We find that HAT is able to match the best possible performance of the Grounded Action Transform (GAT) on this domain, and perform better than (GAT) for some training datasets.

2 Introduction

Skills that enable a robot to perform tasks like walking and kicking are usually hand-coded. It is often prohibitively expensive, both in terms of time and money, to use machine learning to improve upon such hand-coded skills by exploring possibilities on
the robot in the physical world. Robots often require a human present in order to reset it to its original state after each trial of some skill. We are constrained by real world physics in how quickly we can evaluate any particular skill. Robots overheat during trials, which can affect performance and thus accurate measurement of a skill’s success. In some cases, like that of a bipedal robot, the robot falling over because of a bad trial can harm the robot.

In such situations, there is often a simulator available which imperfectly models the robot, and efforts are made to learn good skills in the simulator. In practice, such optimization often overfits to the imperfections in the simulation and the learned skills do not work well in the real world. This phenomenon is referred to as the “reality gap.”

The robot and the simulation are an example of a more general scenario, where we have a “source environment” and a “target environment,” and we wish to teach an agent to perform well in the target environment by training it in the source environment. For the rest of this work, we will refer to the source environment as “simulation” or $E_{\text{sim}}$ and target environment as “reality” or $E_{\text{real}}$.

Consider two skills for completing some task, Skill A and Skill B. Say that Skill A performs better than Skill B in simulation, but Skill B performs better than Skill A in reality. We call this kind of discrepancy, made formal in Section 3, discordance. Discordance can lead an agent to learn a skill which is optimal in the simulation environment, but suboptimal in real environment. In this work, we propose Holistic Action Transform (HAT), an algorithm to minimize differences between simulation and reality which lets us encode these differences in terms of properties of sets of skills. Such an encoding enables us to include ideas like discordance into the training objective for minimizing differences between the two environments.

3 Notations and Definitions

We will frame the problem of skill learning as a Reinforcement Learning problem [11]. An agent starts off at an initial state, $s_0 \in S$, where $S$ is the set of all possible states in the agent’s environment, $E$. There exists a set of all possible actions, $A$, that the agent can take. The agent has a policy, $\pi : S \rightarrow A$. At each discrete timestep $t$, the agent takes an action $a_t \in A$, where $a_t = \pi(s_t)$. The environment then decides the next state of the agent, $s_{t+1}$ according to its dynamics function, $P : S \times A \rightarrow S$. In this work, we work with two environments, $E_{\text{sim}}$ and $E_{\text{real}}$, which share the state and the action space, $S$ and $A$, but have different dynamics functions, $P_{\text{sim}}$ and $P_{\text{real}}$. The agent has access to its current state, but not to the dynamics function.

Each policy, $\pi$ is parameterized by a vector $\theta$. We can modify $\theta$ in order to, in effect, construct a different policy. Thus, for the rest of this work, we will refer to $\theta$ as the “policy.”

An agent acting in an environment according to some $\theta$ undergoes a series of state transitions. A sequence of these state transitions, $\tau = \{s_0, a_0, s_1, a_1, \ldots, s_n, a_n\}$ is a trajectory. We have a cost function which associates a scalar cost with each trajectory. All policies and dynamics functions are deterministic. Thus, each policy leads to a single trajectory in a given environment and thus has a single cost associated with that environment. Thus, we call the cost of a policy $\theta$ in simulation $J_{\text{sim}}(\theta)$ and in reality
Given an initial policy $\theta_0$, the goal of policy optimization is to find a policy $\theta'$, such that it minimizes $J_{\text{real}}$. Because of the reality gap described in Section 2, a policy obtained by performing policy optimization in $E_{\text{sim}}$ does not always perform well in $E_{\text{real}}$. Precisely, given an initial policy $\theta_0$ and a policy obtained through policy optimization in $E_{\text{sim}}$, $\theta'$, such that $J_{\text{sim}}(\theta') > J_{\text{sim}}(\theta_0)$, it is not always true that $J_{\text{real}}(\theta') > J_{\text{real}}(\theta_0)$. In practice, it is often the case that $J_{\text{real}}(\theta') < J_{\text{real}}(\theta_0)$. If the ordering between $\theta_0$ and $\theta_1$ is not the same according to $J_{\text{sim}}$ and $J_{\text{real}}$, we say that $\theta_0$ and $\theta_1$ are discordant. We call this phenomenon discordance.

It is straightforward to see how this discordance can mean that executing policy optimization in $E_{\text{sim}}$ often does not lead to improvement in $E_{\text{real}}$. The goal of this work is to find a method of executing policy optimization in $E_{\text{sim}}$ and having the policy transfer over well to $E_{\text{real}}$. This form of policy optimization is useful because observing $J_{\text{real}}$ is usually much cheaper than observing $J_{\text{sim}}$ because of reasons mentioned in Section 2.

4 Grounded Simulation Learning

In this section, we introduce a simplified version of the Grounded Simulation Learning (GSL) framework, originally described by Farchy et al [3]. The goal of GSL is to improve policy optimization by modifying or grounding the simulator to make it more realistic. While GSL is general enough to support stochastic environments and policies, we make the simplifying assumption here that the environments and policies under consideration are deterministic. GSL makes the following assumptions:

1. $P_{\text{sim}}$ is parameterized by a vector $\phi$ which can be modified in order to render, in effect, a different simulator. We denote $P_{\text{sim}}$ parameterized by $\phi$ as $P_{\phi}$.

2. Given a policy $\theta$, we can record trajectories $\theta$ induces in $E_{\text{real}}$ and $E_{\text{sim}}$. We can also record $J_{\text{real}}(\theta)$ and $J_{\text{sim}}(\theta)$.

3. We have a policy optimization routine, PolicyOptimize, that searches for a $\theta$ that reduces $J_{\text{sim}}(\theta)$. PolicyOptimize also returns a set of candidate policies, $C$, which will be evaluated in $E_{\text{real}}$ in order to determine the best policy.

Say we have a set of policies $\Theta$ and a corresponding dataset, $\mathcal{D} = \{(\theta_0, \tau_{\text{sim}}^0, \tau_{\text{real}}^0), (\theta_1, \tau_{\text{sim}}^1, \tau_{\text{real}}^1), \ldots (\theta_n, \tau_{\text{sim}}^n, \tau_{\text{real}}^n)\}$. Here, $\tau_{\text{real}}^i$ is recorded in $E_{\text{real}}$ and $\tau_{\text{sim}}^i$ is recorded in $E_{\text{sim}}$ respectively. Both are induced by $\theta_i$ in the respective environments. Say $h : S \times S \to \mathbb{R}_{\geq 0}$ is a function which measures the difference between two states, and $s \in \tau_{\text{real}}^i$. $\phi^*$ is a set of parameters such that

$$
\phi^* = \arg\min_{\phi} E[h(P_{\text{real}}(s, \theta(s)), P_{\phi}(s, \theta(s)))]
$$

(1)

Say we have some initial policy $\theta_0$. Within the GSL framework, we carry out the following steps iteratively:

1. Execute $\theta_0$ in $E_{\text{real}}$ to collect the dataset of trajectories, $\mathcal{D}$, in $E_{\text{sim}}$ and $E_{\text{real}}$. 

2. Ground the simulator by solving 1 and setting the parameters for $P_{\text{sim}}$ to $\phi^\star$.

3. Run PolicyOptimize in order to find a set of candidate policies, $C$.

4. Evaluate all policies in $C$ in $E_{\text{real}}$. Choose the policy which has the minimum $J_{\text{real}}$ and designate it as $\theta_1$. Add the trajectories that were obtained while evaluating $C$ in $E_{\text{real}}$ and $E_{\text{sim}}$ to $D$.

Here, the goal of GSL is that $J_{\text{real}}(\theta_1) < J_{\text{sim}}(\theta_0)$. This process can be run iteratively to achieve better performance according to $J_{\text{real}}$.

5 **Grounded Action Transform**

Hanna and Stone introduced Grounded Action Transform (GAT) [4], which improves upon step 2 (grounding) of GSL. This work serves as the primary inspiration for HAT. In the GAT setting, $\phi$ encodes two transformations:

1. The **forward dynamics model**, $g_f : S \times A \rightarrow S$, approximates $P_{\text{real}}$. Thus, in the context of $E_{\text{sim}}$, $g_f$ takes the current state, $s_{\text{curr}}$, and the current action, $a_{\text{curr}}$ and predicts what the next state would be in $E_{\text{real}}$ according to $P_{\text{real}}$. This predicted state is denoted as $s_{\text{pred}}$.

2. The **inverse dynamics model**, $g_i : S \times S \rightarrow A$, is an inverse kinematics model for $E_{\text{sim}}$. The inverse kinematics model, $g_i$, takes $s_{\text{curr}}$ and the predicted next state, $s_{\text{pred}}$, and outputs the predicted action, $a_{\text{pred}}$ that the agent can take to arrive at $s_{\text{pred}}$.

The forward kinematics model, $g_f$ attempts to answer the question, “Given $s_{\text{curr}}$ and $a_{\text{curr}}$ in $E_{\text{sim}}$, what would the next state be in $E_{\text{real}}$ if $a_{\text{curr}}$ were executed from $s_{\text{curr}}$?”

On the other hand, the inverse kinematics model, $g_i$, attempts to answer the question, “What action should the agent take in $E_{\text{sim}}$, in order to achieve the state achieved by some agent which executes $a_{\text{curr}}$ from $s_{\text{curr}}$ in $E_{\text{real}}$?” When the agent passes $a_{\text{curr}}$ to $E_{\text{sim}}$, the environment applies the transformation $g_i(s_{\text{curr}}, g_f(s_{\text{curr}}, a_{\text{curr}}))$ before applying the dynamics function, $P_{\text{sim}}$. This transformation is an instance of an action modifier function, which modifies the action output by the agent’s policy into a different action in order to improve the fidelity of $E_{\text{sim}}$ vis-a-vis $E_{\text{real}}$.

An issue with this approach is that $g_i$ and $g_f$ minimizes the difference between $E_{\text{sim}}$ and $E_{\text{real}}$ at a single timestep granularity. Any remaining difference has the potential to compound over timesteps. Thus, given a $\theta$, the agent in $E_{\text{sim}}$ might gradually veer away from the trajectory that would have been observed in $E_{\text{real}}$. In addition, this optimization objective does not directly capture the issue of discordance as described in section 3. It only captures the difference between the two environments at a single timestep granularity.

6 **Holistic Action Transformation**

In this section, we introduce Holistic Action Transform (HAT), the principle algorithmic contribution of this work. HAT builds upon the grounding step (step 2) of the GAT
Algorithm 1 Holistic Action Transform (HAT) pseudocode. Inputs: An initial policy, \( \theta_0 \), an environment \( E_{real} \), an environment \( E_{sim} \), and a function \( d \) which gives a scalar loss for a dataset consisting of policies and their corresponding trajectories in \( E_{sim} \) and \( E_{real} \). GenerateTrainingSet generates a set of policies \( \Theta \). Rollout evaluates \( \Theta \) in \( E_{sim} \) and \( E_{real} \) augmented by \( f \). Rollout then returns a dataset of the form \( \{(\theta_1, \tau_{sim}^1, \tau_{real}^1), (\theta_2, \tau_{sim}^2, \tau_{real}^2), \ldots, (\theta_n, \tau_{sim}^n, \tau_{real}^n)\} \). This dataset is used to train a function \( f : S \times A \rightarrow A \) such that \( d \) is minimized for policies in \( \Theta \), trajectories \( \tau \). During the policy optimization step, optimize, and at each timestep \( t \), \( s_{t+1} \) is calculated as \( a_{t+1} = P_{sim}(s_t, f(s_t, a_t)) \) instead of \( P_{sim}(s_t, a_t) \).

1: function GAT
2: \( \Theta \leftarrow \text{GenerateTrainingSet}(\theta_0) \)
3: \( \text{cost}(f) \leftarrow d(\text{Rollout}(E_{real}, E_{sim}, f, \Theta)) \)
4: \( f^* \leftarrow \text{argmin}_f \text{cost}(f) \)
5: \( C \leftarrow \text{PolicyOptimize}(E_{sim}, \theta_0, f^*) \)
6: \( \text{return argmin}_{\theta \in C} J(\theta) \)
7: end function

Algorithm 2 Rollout pseudocode. Inputs: Environments \( E_{real} \) and \( E_{sim} \), an action modifier function \( f : S \times A \rightarrow A \) and a set of policies \( \Theta \). The \( \text{EvalInEnvironment} \) function returns the trajectory of the given policy in the given environment.

1: function Rollout
2: \( D \leftarrow \{\} \)
3: for \( \theta \) in \( \Theta \) do
4: \( \tau_{real}^i \leftarrow \text{EvalInEnvironment}(\theta, E_{real}) \)
5: \( \theta'(s) \leftarrow f(s, \theta(s)) \)
6: \( \tau_{sim}^i \leftarrow \text{EvalInEnvironment}(\theta', E_{sim}) \)
7: \( D \leftarrow \text{append}(D, (\theta_i, \tau_{real}^i, \tau_{sim}^i)) \)
8: end for
9: \( \text{return } D \)
10: end function

framework.

In the HAT setting, \( \phi \) encodes the transformation \( f : S \times A \rightarrow A \). At each timestep \( t \), we pass \( f(s_t, a_t) \) to \( P_{sim} \) instead of passing \( a_t \) to \( P_{sim} \). Note that in the GAT setting, \( f(s_t, a_t) = g_t(s_t, g(s_t, a_t)) \).

Say we have a dataset \( D \), similar to the dataset described in Section 4. We denote a function which takes such a dataset and returns a scalar cost as \( d \). Instead of finding an action transformation that minimizes the one step loss between subsequent states (refer to Section 5), HAT directly finds an action transformation function \( f \) so that \( d \) is minimized. Examples of \( d \) which can be used are:

1. **Mean Absolute Error**: Say \( \tau_{sim}^0 \) and \( \tau_{real}^1 \) are two trajectories induced by the same policy in \( E_{real} \) and \( E_{sim} \). Say \( \tau_{sim}^0 = \{s_{sim}^0, a_{sim}^1, \ldots, s_{sim}^m, a_{sim}^m\} \) and \( \tau_{real}^1 = \{s_{real}^0, a_{real}^1, \ldots, s_{real}^m, a_{real}^m\} \). The Mean Absolute Error between \( \tau_{sim}^0 \) and \( \tau_{real}^1 \) is defined as \( \text{MAE}(\tau_{sim}^0, \tau_{real}^1) = \sum_{i=0}^{m} |s_{i}^{real} - s_{i}^{sim}| \). In this instantiation of \( d \), it is defined as \( d(D) = \sum \text{MAE}(\tau_{sim}^0, \tau_{real}^1); (\theta, \tau_{sim}^0, \tau_{real}^1) \in D \).
2. **Concordance**: Two policies, $\theta_i$ and $\theta_j$, are concordant if $J_{\text{sim}}(\theta_i) > J_{\text{sim}}(\theta_j) \iff J_{\text{real}}(\theta_i) > J_{\text{real}}(\theta_j)$. Similarly, we say $\theta_i$ and $\theta_j$ are discordant if this condition is false. For a set of policies $\Theta$, concordance is defined as:

$$\text{concordance}(\Theta) = \frac{\#\text{Concordant pairs of policies} - \#\text{Discordant pairs of policies}}{\#\text{Total pairs of policies}}.$$ 

Clearly, if all pairs of policies are concordant, this value will be equal to 1, while if they are all discordant, it will equal -1. In this setting, $d(\Theta) = -\text{concordance}(\{\theta \mid (\theta, \tau_{\text{sim}}, \tau_{\text{real}}) \in D\})$

Mean Absolute Error captures the idea of trajectories in $E_{\text{sim}}$ and $E_{\text{real}}$ being “similar.” Concordance captures the issue of disconcordance that was described in Section 3. Say we are using a policy-sampling method like CMA-ES [5] to perform PolicyOptimize. Say $\theta^*$ is the policy obtained by executing PolicyOptimize directly in $E_{\text{real}}$. We show in Appendix A that if all pairs of policies are concordant in $E_{\text{sim}}$ and $E_{\text{real}}$, we can obtain $\theta^*$ by performing policy optimization directly in $E_{\text{sim}}$.

We show pseudocode for HAT in Algorithm 1. The basic approach is to first find an action modifier function $f$ which minimizes $d$, and then run the PolicyOptimize function to find a set of candidate policies $C$. Similarly to GAT, policies in $C$ are then evaluated in $E_{\text{real}}$.

### 7 Results

In this section, we describe the Linear Dynamical System domain in which we run our experiments, and compare HAT and GAT based on $J_{\text{real}}$ for policies learned using these two frameworks.

#### 7.1 Experimental Setup

We evaluate HAT on the *Linear Dynamical System* domain, illustrated in Fig 1. In this work, the state of the agent is fully determined by a vector of length 4. A policy, $\theta$, is
an affine transformation agent’s current state. The dynamics function associated with $E_{real}$ is of the form $P_{real}(S_t, A_t) = k_{real} \ast S_t + l_{real} \ast A_t$, where $k_{real}$ and $l_{real}$ are both linear transformations and features of the environment. The dynamics function associated with $E_{sim}$ is of the form $P_{sim} = k_{sim} \ast S_t + l_{sim} \ast A_t$, created by perturbing $k_{real}$ and $l_{real}$. The parameters of the dynamics functions, that is, $k_{real}$, $l_{real}$, $k_{sim}$ and $l_{sim}$, are unknown to the agent.

The agent starts off at an initial position of $(0 0 0 0)$, and its goal is to reach $(10 10 10 10)$. Each policy is evaluated for 10 timesteps. Thus, each trajectory is of length 10. For a policy $\theta$ which induces the trajectory $\{s_0, a_0, s_1, a_1, \ldots, s_9, a_9\}$, $J(\theta) = \sum_{i=0}^{9} \|s_i - (10 10 10 10)\| - \|a_i\|$. While $k_{sim}$, $l_{sim}$, $k_{real}$ and $l_{real}$ were fixed for all our experiments, we made the choice based on the following criteria:

1. We ran policy optimization using CMA-ES in $E_{real}$ for 10 generations with a population size of 150. This gives us 10 sets of 150 policies each, with one set per generation. We measured the concordance of these sets of policies. We aimed to choose $k_{sim}$ and $l_{sim}$ so that the concordance within a generation was between 0.5 and 0.8. This heuristic is visualized in Figure 2.

2. We ran policy optimization using CMA-ES directly for 100 generations with a population size of 150 in $E_{real}$. We observed the best policy achieved. We ensured $J_{real}$ for this policy was better than $J_{real}$ for the policy obtained by running CMA-ES in $E_{sim}$. This heuristic is visualized in Figure 3.

The range of concordance means that there is a reality gap (refer Section 2) between $E_{real}$ and $E_{sim}$, but there is still correlation between $J_{real}$ and $J_{sim}$. We see in our experiments below that this discordance leads to policy optimization in $E_{sim}$ converging to a different, less optimal point than the point policy optimization in $E_{real}$ converges to.

We use CMA-ES as the PolicyOptimize routine. The set of candidate policies $C$ to be evaluated in $E_{real}$ is obtained by selecting policies with the lowest $J_{sim}$ within their generation.

**Baselines** We use two baselines to compare HAT against.

- **NoMod** In this setting, we do not use any action modifier. We simply run policy optimization in $E_{sim}$.

- **GroundTruthGAT** In the Linear Dynamical System domain, the best GAT forward and inverse dynamics models can be analytically computed. These ideal forward and inverse dynamics models are used in this baseline, and represents the best possible performance of GAT.

**Training HAT modifiers** For each experiment, we do the following:

1. **Creating $\Theta$** We ran policy optimization using CMA-ES for three generations with a population size of 150, for a total of 450 policies in the training set. We recorded the trajectories in $E_{sim}$ and $E_{real}$ as well as $J_{sim}$ and $J_{real}$ for each policy in $\Theta$. 

7
Figure 2: Visualizing Concordance between $E_{sim}$ and $E_{real}$. We ran PolicyOptimize in $E_{real}$ for 10 generations with a population size of 150. From the set of policies evaluated during PolicyOptimize, we then created 10 sets of 150 policies each, one set per generation. We measured concordance for each set of policies, and the concordance for each generation is visualized below. We see that there is originally high concordance, but as the agent learns an optimal policy for $E_{real}$, differences between the two environments become more prominent and concordance goes down.

2. **Optimizing $\phi$:** We train the action modifier, $f$, parameterized by $\phi$, using CMA-ES. We use 300 generations of population size 150 each, and choose the best action modifier according to $d$.

3. **Optimizing $\theta$:** We used this action modifier while executing PolicyOptimize, using CMA-ES again. For policy optimization, we used 100 generations with a population size of 150.

4. Repeat steps (1), (2), and (3) five times with random seeds shared between experiments. We also did an experiment where we fixed the dataset we trained on, and thus only ran (2) and (3) multiple times.

**HAT Training Objectives**  We experimented with the following settings of $d$ to train HAT.

1. Mean Absolute Error (HAT-MAE): We set $d$ to the MAE function as described in Section 6.

2. HAT-Joint: We set $d$ to a linear combination of MAE and concordance. Both are described in Section 6. In particular, we set $d(\Theta) = -\text{MAE}(\Theta) + 10,000 *$
Figure 3: This figure visualizes the reality gap between $E_{\text{sim}}$ and $E_{\text{real}}$. We ran PolicyOptimize in both environments. We then choose the best policy from each generation according to the cost functions in their respective environments. Finally, we plot the $J_{\text{real}}$ for policies chosen this way. We see that $J_{\text{real}}$ for policies obtained while optimizing for $J_{\text{real}}$ is lower than the $J_{\text{real}}$ for policies obtained while optimizing for $J_{\text{sim}}$. This suggests a reality gap between the two environments.

\[ J_{\text{real}} \text{ vs Num Generations} \]

---

concordance$(\Theta)$. The factor of 10,000 accounts for the difference in scale between Concordance and MAE.

We briefly experimented with using only concordance as the optimization objective for simulator optimization. However, based on our experiments, using concordance by itself is insufficient. We found that, while HAT was able to learn an action modifier that induced high concordance on the training set, the modifier did not generalize well to policies not in the training set. Policies learned when using an action modifier trained on only concordance do not perform as well in the real world as policies learned when using action modifiers trained on the objectives described above.

7.2 Numerical Results

In Figure 4, we plot the $J_{\text{real}}$ of a policy learned after one iteration of HAT. In this plot, the dataset used to train HAT was randomly generated but is fixed across five trials. We saw that the average cost of a policy learned using HAT is statistically better than one learned using GAT, and the p-value calculated using Paired Student’s T-Test is less than 0.0001. The difference between the MAE-only HAT optimization and Joint HAT optimization (which includes concordance as an optimization criterion) also shows that optimizing for concordance improves policy optimization for this training data.
Figure 4: Cost of transferred policy. We re-run the policies in $C$ in $E_{\text{real}}$. The datasets on which HAT is trained are randomly generated and fixed across trials.

In Figure 5, the data for NoMod and GroundTruthGAT is the same as that in Figure 4. For the HAT-Joint bar, we carried out ten trials, and for each trial we trained HAT on a different randomly generated dataset. Also for each trial, we ran PolicyOptimize 5 times with different seeds. We saw that while HAT performs slightly better than GAT, this difference is not statistically significant. Given these two plots, it seems as if the choice of training dataset can have an impact on the performance of HAT when using the Joint training objective, and it raises an interesting questions about what makes for a good dataset to train a HAT modifier.

Since it is expensive to run robots for long periods of time, data efficiency is an important issue for sim2real transfer. We analyzed whether our method of choosing policies to train HAT modifier is data efficient. Our findings are visualized in Figure 6. We analyzed the performance of HAT when it is trained on fewer policies. For each bar on the graph, we randomly created 5 datasets of the corresponding sizes and ran simulation optimization and policy optimization once. We show the average $J_{\text{real}}$ for the final policy obtained through this method. We saw that we were not able to beat GroundTruthGAT even when using 120 policies, which is already a lot of policies to evaluate on a real robot. The fact HAT does not yet allow learning policies in a data efficient manner suggests that there is potential to choose policies to train HAT in a better way than we did in this work. We leave finding better sampling methods for future work.

We are also interested in whether there is a correlation between $J_{\text{real}}$ for a policy learned through HAT, and the concordance induced by the same HAT modifier between $E_{\text{real}}$ and $E_{\text{sim}}$. In Figure 7 we plot the trials ranked according to concordance on datasets trained on other trials vs. them ranked according to performance of policies in $E_{\text{real}}$. The concordance between these two rankings is 0.51. We also analyzed the raw
Figure 5: Cost of transferred policy. We re-run the policies in $C$ in $E_{\text{real}}$. We randomly generated ten datasets to train a HAT modifier. We did five trials of PolicyOptimize on each dataset, for a total of 50 trials. We did five trials of NoMod and GroundTruth-GAT.

![Graph showing cost of theta1 in E_real, varied datasets.]

Values corresponding to these rankings (in Appendix). In that comparison, Pearson’s R value is 0.77 with an R-squared value of 0.6. This correlation is not perfect, and that might be because of the stochasticity of the CMA-ES as a PolicyOptimize routine. Another possibility is that some modifiers reduce discordance in the part of the policy space that the training dataset was sampled from, but might not reduce discordance in other parts of the policy space explored by CMA-ES during policy optimization.

It is interesting to see how quickly HAT can learn a policy which performs well in $E_{\text{real}}$. In Figure 8, we visualize the relative performance of NoMod, GroundTruthGAT and HAT over generations. This was done by training ten HAT modifiers on ten different randomly generated datasets. We then ran PolicyOptimize once per HAT modifier, and also ten times each in the GroundTruthGAT modifier and NoMod settings. Each run of PolicyOptimize used CMA-ES as the underlying optimization algorithm with 100 generations and a population size of 150. The set of candidate policies to be executed in $E_{\text{real}}$, $C$, was created by choosing the policy with the lowest $J_{\text{sim}}$ in each generation, so that we had 10 sets of candidate policies for each method. Finally, we ranked each generation of CMA-ES according to $J_{\text{real}}$ of the policy from that generation in $C$.

We hypothesized that HAT will initially learn faster than other methods since our training dataset is sampled from a similar distribution to the ones the first few steps of PolicyOptimize sample from. However, as Figure 8 shows, all methods achieve their best performance in $E_{\text{real}}$ around the same generation (60). We think this might be because the training set of policies was sampled from the first three generations.
Figure 6: For each listed size, we create 5 randomly generated datasets of that size to train HAT. We then run PolicyOptimize for each instantiation of HAT. Here, we visualize the average $J_{\text{real}}$ for the final policy chosen after policy optimization.

![Cost of theta; in $E_{\text{real}}$, varied sizes of datasets.](image)

of PolicyOptimize, and concordance might drop off quickly for the rest of the generations. For future work, we think it will be interesting to explore other ways of generating the training set for HAT. For example, another way of sampling policies is to run PolicyOptimize in $E_{\text{sim}}$, and sample some policies from each generation rather than only sampling from within the first three generations.
Figure 7: We trained HAT modifiers on 10 different randomly generated datasets. We then ran five trials of PolicyOptimize in $E_{\text{sim}}$ for each of the ten HAT modifier. Finally, we ranked the HAT modifiers according to the average concordance they induce on the datasets used to train the ten modifiers, and also according to average $J_{\text{real}}$ of the policies learned during the 5 trials of PolicyOptimize. We see below that the two rankings have positive correlation. The concordance between the two rankings is 0.51.
Figure 8: We trained HAT modifiers on 10 different randomly generated datasets. We then ran one trial of PolicyOptimize in $E_{\text{sim}}$, using CMA-ES as the underlying optimization algorithm, for each of the ten HAT modifiers. For each generation, we chose the policy with the lowest $J_{\text{sim}}$. We then ranked the generations against each other according to $J_{\text{real}}$ for these policies. These rankings are visualized in this graph. We saw that all three methods usually reach their best performance between generations 30 and 50, after which performance starts degrading. After this, performance degrades more in the case of HAT than in the other two cases, which suggests that HAT does not always generalize well to policies not in the training set. Thus, there is room for experimenting with how the training data is chosen for HAT.
8 Related Works

Transferring skills learned in simulation to reality is valuable for the reasons mentioned above, and thus is an active area of research. One way to do this skill transfer is to make the simulation environment more accurate by making direct measurements about the real environment and incorporating these measurements into the simulator. In order to teach a quadrapedal robot how to walk, Tan et al. took the robot apart, weighed every part, and incorporated information about inertia of every part into the simulation [12] prior to carrying out policy optimization in simulation. This approach saw considerable success, but is less general than a learning-based approach.

Reidmiller et al. experimented with teaching a physical robotic arm to complete tasks like grabbing and stacking by executing policy optimization directly on the physical arm [8]. However, in order to do this successfully in a reasonable amount of time, they had to simplify the interface used by the program to interact with the robotic arm. In addition, such an approach is also not feasible on many other kinds of robots, like bipedal robots, because of reasons outlined in Section 2.

Rusu et al. [9] [10] introduced the progressive network architecture, which allows the agent in \( E_{\text{real}} \) to better remember what was learned in simulation \( E_{\text{sim}} \). This experience learnt in simulation aids in fine tuning the agent in \( E_{\text{real}} \). Thus, a portion of policy optimization is carried out in \( E_{\text{real}} \). In contrast, HAT allows us to carry out all policy optimization in \( E_{\text{sim}} \).

Chebotar et al. used iterative domain randomization to teach a robotic arm to complete tasks like opening drawers and swinging a peg on a string into a hole [1]. Their method iteratively creates an ensemble of simulations which approximate reality. The simulations are sampled from a distribution created according to measured differences in trajectories in simulation and reality. They focus on optimizing simulation parameters like friction, mass coefficient parameters etc. rather than learning abstract dynamics of the environment. We think that some of the ideas mentioned here in our work, especially concordance (Section 6), can be combined with this work to inform this change in distributions over iterations.

Mehta et al. [7] and Zakharov et al. [14] both devised a method to automatically create a set of simulations in which the policy used by the agent performs poorly. This is in service of learning a stable policy, ie. one which can perform well when there is high variance in various properties of the environment. In both works, there was an observation that the distribution from which this set is sampled can have an impact on the stability of the learned policy. It might be interesting to place a “minimum concordance” constraint between the real environment and the simulators as a reasonable constraint on the space of possible simulators explored.

Clavera, Ignasi, et al. experimented with learning a policy which can easily adapt to an ensemble of simulators with a single policy gradient step [2]. They find success in training agents to perform well in the Mujoco domain [13]. For this method, the way in which differences between simulation and reality is measured must be a differentiable function with respect to simulation parameters. However, this rules out some of the measures we describe later, like Concordance (Section 6), which are not even continuous functions.

James et al. treated the Simulation to Reality transfer problem as a sensor augmen-
They were able to double the success rate of a policy optimization method in picking up a previously unseen physical object with a robotic arm. They achieved this increase in success rate by first training a neural network to convert an image of a robotic arm to a canonical representation. The training set of images was recorded in simulation and augmented in various ways. This manner of sensor augmentation contrasts with our method of actuator augmentation, and it is interesting to study how one system might impact the other if both systems are in place.

9 Future Work

This work raises some interesting questions and also has limitations, which we discuss here. We saw that the value of the best policy produced by HAT was quite stable over multiple trials when using the same dataset to train. However, this stability was not observed once we changed the datasets for each trial. This raises the question, what makes a dataset good to train HAT on?

Depending on the training objective HAT requires more runs of the simulator during training than GAT. If we are using concordance as a training objective, CMA-ES as the PolicyOptimize routine, and our training set, Θ, has n policies, HAT will require n times more runs of the simulator than GAT does. It will be interesting to explore if we can get similar performance if we estimate the d with fewer runs of the simulator rather than evaluating the individual on every single policy in the Θ.

We have used a simple domain to test HAT. This decision was made because it enabled us to experiment with different versions of HAT algorithm quickly. However, the kind of noise we might encounter when trying this method in the real world robot will be different that the one seen in this toy domain. That kind of noise might also make the intuition of optimizing for concordance more pertinent (Appendix B). Thus we would like to test HAT in real world domains.

We also think that it will be interesting to combine this work with some of the previous work described earlier. For example, we could try to train the forward and inverse dynamics models introduced by Hanna and Stone [4] and add concordance as a loss to the simulation training objective.

10 Conclusion

In this work, we have proposed a new learning algorithm, HAT, which, enables the programmer to specify the simulation training objective in terms of properties of sets of trajectories. We have shown that concordance, a measure of similarity between $E_{\text{sim}}$ and $E_{\text{real}}$ which can be expressed in this way, is correlated with the quality of the final solution that is output when working with a simple environment, Linear Dynamical System.
11 Acknowledgments

I would like to thank Josiah Hanna and Dr. Peter Stone for their invaluable advice and guidance throughout this project. Thank you to Dr. Scott Niekum and Dr. Alan Cline for serving on my committee. I would also like to thank Megan Lamonica for providing feedback on previous versions of this work. In no particular order, I would like to thank Sanmit Narvekar, Scott Cao, Brahma Pavse, Seth Taylor, Saket Sadani, and everyone else with whom I have discussed this project. Finally, I would like to thank Dr. Calvin Lin for building and supporting the Turing Scholars program, from which I have benefited immensely.
References


A Showing that Perfect Concordance Closes Reality Gap

We define the following predicates:

- **Optimal**: Say we have an initial policy \( \theta_0 \). Say we perform the policy optimization in \( E_{\text{sim}} \) and get a policy \( \theta_1^{\text{sim}} \), and also perform policy optimization in \( E_{\text{real}} \) and get \( \theta_1^{\text{real}} \). We can expect \( \theta_1^{\text{sim}} \) to perform as well in \( E_{\text{real}} \) as \( \theta_1^{\text{real}} \) performs in \( E_{\text{real}} \).

- **Concordant**: For all policies \( \theta_i \) and \( \theta_j \) sampled during policy optimization, \( \theta_i \) and \( \theta_j \) are concordant.

We also make the following assumptions:

1. CMA-ES or a similar policy-sampling based approach which iteratively samples policies and makes decisions about the next set of policies to be sampled based on their ranking is used as the PolicyOptimize routine. [1]

2. The policies and the dynamics systems of the environments are deterministic.

We argue that Concordant \( \implies \) Optimal. Say we run PolicyOptimize with the underlying algorithm of CMA-ES directly in \( E_{\text{real}} \). We call this Run A. We use the seed policy \( \theta_0 \). Say PolicyOptimize returns the policy \( \theta^* \) which performs optimally in \( E_{\text{real}} \). Say we then run PolicyOptimize in \( E_{\text{sim}} \) with the same seed policy. We call this Run B.

Proving that the same set of policies is sampled in Run A and Run B First, since we are using the same seed for both runs, the first generation of policies sampled will be the same between the two runs. Second, let’s assume that for the \( k \)th generation, the set of policies sampled for the two runs is the same. From assumption [1], all policies in \( k \)th generation are concordant and will thus be ranked exactly the same by \( J_{\text{real}} \) and \( J_{\text{sim}} \). Because the way in which CMA-ES samples policies for the \( (k + 1) \)th generation depends only on how policies in the \( k \)th generation are ranked, policies in the \( (k + 1) \)th generation will also be the same between the two runs of CMA-ES. By the principle of mathematical induction, the set of policies is sampled in both runs.

Proving that the the best policy sampled in Run B is \( \theta^* \) Let’s assume that the best policy sampled in Run B is \( \theta' \neq \theta^* \). We have already proved that \( \theta^* \) is sampled during Run B. If the best policy sampled during Run B is \( \theta' \), that means that \( J_{\text{sim}}(\theta') > J_{\text{real}}(\theta') \). However, since \( \theta^* \) is the best policy sampled during Run A and \( \theta' \) is also sampled during Run A, \( J_{\text{sim}}(\theta^*) > J_{\text{real}}(\theta') \). However, since we assume that the set of policies sampled during execution of Run A are concordant, this leads to a contradiction. Therefore, the best policy in Run B is \( \theta^* \).

B Why Mean Absolute Difference Is Not Enough

In this section, we would like to explain the intuition for why, in general, eliminating the Mean Absolute Difference between trajectories in \( E_{\text{real}} \) and \( E_{\text{sim}} \) is not enough to bridge the reality gap.
Consider the task of a robotic arm opening a drawer. The cost function, $J$, is based on how quickly the robot opens the drawer. Say that the trajectory of the robotic arm is captured by its various joint angles. Say that we have an imperfect simulator available, and that we are able to learn an action modifier that is able to eliminate the MAE between the robots in the two environments.

Now, consider the case in which the drawer in simulation is shifted a few centimeters to the left of where it is placed in the real world. Say $\theta^\star$ is the optimal policy for $E_{\text{real}}$. Even though the simulation robot will be able to follow the same trajectory as the physical robot for $\theta^\star$, the simulation robot will miss the drawer handle on its first try. On the other hand, a policy $\theta'$, which is optimal in simulation, will miss the drawer handle in reality on its first try. We conjecture, but don’t prove, that including concordance in the simulation optimization objective could induce a mapping between $\theta^\star$ and $\theta'$, which can enable the robot to learn $\theta^\star$ in $E_{\text{sim}}$.

**C Data**

Here, we provide data for the correlation analysis visualized in Figure 7.

Table 1: For each trial, we sampled a different training of policies. We then ran policy optimization in $E_{\text{sim}}$ five times for each trial. Here, we show the average $J_{\text{real}}$ for each trial, and the average number of discordant pairs when each trial is evaluated on training sets for all 10 trials.

<table>
<thead>
<tr>
<th>Trial Number</th>
<th>Average value in $E_{\text{real}}$</th>
<th>Average Number of discordant pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66.65</td>
<td>6103.8</td>
</tr>
<tr>
<td>2</td>
<td>68.80</td>
<td>6077.9</td>
</tr>
<tr>
<td>3</td>
<td>66.03</td>
<td>6082.5</td>
</tr>
<tr>
<td>4</td>
<td>74.75</td>
<td>6385.5</td>
</tr>
<tr>
<td>5</td>
<td>69.89</td>
<td>6124.3</td>
</tr>
<tr>
<td>6</td>
<td>68.76</td>
<td>6107.1</td>
</tr>
<tr>
<td>7</td>
<td>66.76</td>
<td>6190.9</td>
</tr>
<tr>
<td>8</td>
<td>70.07</td>
<td>6394.3</td>
</tr>
<tr>
<td>9</td>
<td>71.38</td>
<td>6104.6</td>
</tr>
<tr>
<td>10</td>
<td>75.53</td>
<td>6678.3</td>
</tr>
</tbody>
</table>