RIDM: Reinforced Inverse Dynamics Modeling for Learning from a Single Observed Demonstration

Brahma S. Pavse [1] Faraz Torabi [1] Josiah Hanna [1] Garrett Warnell [2] Peter Stone [1]

Abstract

Imitation learning has long been an approach to alleviate the tractability issues that arise in reinforcement learning. However, most literature makes several assumptions such as access to the expert's actions, availability of many expert demonstrations, and injection of task-specific domain knowledge into the learning process. We propose reinforced inverse dynamics modeling (RIDM), a method of combining reinforcement learning and imitation from observation (IfO) to perform imitation using a single expert demonstration, with no access to the expert's actions, and with little task-specific domain knowledge. Given only a single set of the expert's raw states, such as joint angles in a robot control task, at each time-step, we learn an inverse dynamics model to produce the necessary low-level actions, such as torques, to transition from one state to the next such that the reward from the environment is maximized. We demonstrate that RIDM outperforms other techniques when we apply the same constraints on the other methods on six domains of the MuJoCo simulator and for two different robot soccer tasks for two experts from the RoboCup 3D simulation league on the SimSpark simulator.

1. Introduction

Learning from experience, or reinforcement learning (RL) (Sutton & Barto, 1998), has proven to be an effective approach for enabling artificial agents to execute new behaviors. However, a major limitation is that this learning process can be extremely slow and expensive. This limitation is especially true for physical robots, which must operate in the real world, in real time, and are prone to wear and tear.

[1]University of Texas at Austin, Austin, USA [2]U.S. Army Research Laboratory. Correspondence to: Brahma S. Pavse <brahmasp@cs.utexas.edu>.

Appearing in the Proceedings of the ICML Workshop on Imitation, Intent, and Interaction (I3), Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

In order to alleviate these issues, imitation learning (Schaal, 1997; Argall et al., 2009) techniques have been employed to guide a learning agent along an expert's trajectory to speed up the learning process.

While imitation learning has proven to be very effective, it has often operated under three assumptions. First, the learner often needs access to the expert's actions. This restriction proves to be a very limiting constraint since it prevents us from using many unused resources such as YouTube videos that may not include expert actions. Second, the developed methods often require access to many expert demonstrations. Since expert demonstration collection is often expensive, we would like to reduce our dependence on the availability of many demonstrations. Third, domain knowledge is usually injected in the state space during the learning process. For instance, in the case of an arm robot, in a reaching task where the goal is to get the end effector close to a specific location, the distance from the target location is usually included in the state space. This information is task-specific and often makes the learning process much simpler. However, in general, acquiring this type of knowledge may be expensive; therefore, we would like to distance ourselves from this idealized situation and remove the task-specific domain knowledge used in the state space (in the rest of the paper this is referred to as raw state space).

Here, we propose RIDM, a method of combining reinforcement learning and model-based imitation from observation, to perform imitation of an expert from a *single* expert demonstration, with *no* action information, and with *no* task-specific domain knowledge in the state space. More specifically, given a single set of only the expert's raw states at each time-step, our algorithm uses a randomly-initialized inverse dynamics model to infer actions to transition from the current state to the next. It then executes these actions in the environment. It finally uses the generated data to train the inverse dynamics model such that the cumulative reward from the environment is maximized. This process repeats until convergence. The reward is used for the learner to explore and the state-only expert demonstration is used as a template for ideal behavior. In our experiments, which are focused on robot control domains, we model our inverse dynamics model as a PID controller[1], and are interested in learning the gains of the PID controller to infer the actions. To the best of our knowledge, we are the first to use the PID controller as an inverse dynamics model. We reduce the task-specific domain knowledge in the state space exposed to the learner to only joint angle values per time-step. We use covariance matrix adaptation evolution strategy (CMA-ES) (Hansen et al., 2003) as our RL algorithm to optimize the inverse dynamics model parameters.

The remainder of the paper is organized as follows. Section 2 discuses the current literature in imitation from observation, integrating reinforcement learning and imitation learning, and robot soccer skill learning. Section 3 outlines the preliminaries and background necessary for the remaining content of the paper. Section 4 details our proposed control algorithm, RIDM. Section 5 discusses our experiments on the MuJoCo domain and SimSpark robot soccer simulator. Finally, Section 6 outlines a summary and future work. We also include a Supplementary Materials in Appendix A that includes details of the PID controller and additional experiments.

2. Related Work

This section provides a broad outline of research related to our work. The section is organized as follows. Section 2.1 details previous work on imitation from observation. Section 2.2 discusses efforts in integrating reinforcement learning and imitation learning. Finally, Section 2.3 details successful efforts of using CMA-ES for robot skill learning.

2.1. Imitation from Observation

The focus of imitation from observation (IfO) (Liu et al., 2017; Torabi et al., 2019d) is to learn a policy that results in similar behavior as the expert demonstration with state-only demonstrations. There are broadly two approaches: (1) model-based and (2) model-free. In our work, we are focused on model-based. For details on model-free refer to the work of Merel et al. (2017), Henderson et al. (2017a), Torabi et al. (2019b; 2019c), Stadie et al. (2017), Sermanet et al. (2017), and Dwibedi et al. (2018).

Most model-based IfO algorithms use an inverse dynamics model, i.e., a mapping from state-transitions to actions. The most related work to ours may be the work of Nair et al. (2017a), where they show the learner a single demonstration of an expert performing some task with the intention of the learner replicating the task. They do this by allowing the learner to undergo self-supervision and collect states and actions, which is then used to train a neural network inverse dynamics model. The learned model is then applied on the expert demonstration to infer the expert actions. The actions are then executed to replicate the demonstrated behavior. Another method of this type is behavioral cloning from observation (BCO) by Torabi et al. (2018), which, similarly, first trains an inverse dynamics model in a self-supervised fashion, and applies the learned model on the expert demonstration(s) to infer the expert actions. However, BCO then trains a policy by behavioral cloning (BC) (Pomerleau, 1991), which maps the expert states to the inferred actions.

Our work differs from past work in that we reinforce the learning of an inverse dynamics model by incorporating the provided environment reward.

2.2. Integrating Reinforcement Learning and Imitation Learning

Another area of research related to our work is dealing with the case when an expert demonstration may be a good starting point, but may be sub-optimal. One way to address this issue is by combining reinforcement learning and imitation learning.

There has been significant effort to combine reinforcement learning and imitation learning. For example, Knox & Stone (2010; 2012) introduced the TAMER + RL framework that combines manual feedback with rewards from the MDP. Lakshminarayanan et al. (2016) uses a hybrid formulation of reward and expert state-action information in the replay buffer when training deep Q-network (DQN) to speed-up the training procedure. Hosu & Rebedea (2016) use deep RL to learn an Atari game but they use human checkpoint replays as starting points during the learning process instead of restarting the game at the end of the episode. Subramanian et al. (2016) and Nair et al. (2017b) use IL information to alleviate the exploration process in RL. Hester et al. (2017) pre-train a deep neural network by optimizing a loss that includes a temporal difference (TD) loss as well as supervised learning loss with the expert actions. Zhu et al. (2018) optimize a linear combination of the imitation reward outputted by generative adversarial imitation learning (GAIL) (Ho & Ermon, 2016) and the task reward. However, it is important to note that these works assume that the learner has access to the expert's actions.

Our work is distinct from the current literature in that we focus on the integration of reinforcement learning and imitation from observation where we do *not* have access to expert actions.

2.3. Robot Soccer Skill Learning

There has been much success of using covariance matrix adaptation evolution strategy (CMA-ES) (Hansen et al., 2003) for derivative-free optimization in reinforcement

[1]Refer to the Supplementary Materials section for details about the PID controller.

learning. Salimans et al. (2017) have noted the scalability of evolutionary algorithms for reinforcement learning tasks. We have also seen much success of applying CMA-ES to skill learning in robot soccer (Urieli et al., 2011). For example, for walking, MacAlpine et al. (2012) have used CMA-ES to learn an omnidirectional walk engine, which is currently among the best in the RoboCup 3D simulation league. For kicking, Depinet et al. (2015) develop a method called KSOBI (keyframe sampling, optimization, and behavior integration) that uses CMA-ES to learn a 20m long distance kick.

In our work, we make use of CMA-ES to learn and improve upon the expert's walking and kicking skills.

3. Preliminaries

This section describes the relevant background needed to understand the later sections. In particular, Section 3.1 gives an idea of the machine learning problem we are interested in, and Section 3.2 discusses the basics of imitation learning.

3.1. Reinforcement Learning (RL)

We model agents interacting in some environment as a Markov decision process (MDP). An MDP is denoted by the tuple $M = \langle S, A, T, R, \gamma \rangle$, where S is the state space of the agent, A is the action space of the agent, T are the transition probabilities of moving from one state to another given the agent took a particular action i.e. $T : S \times A \times S \rightarrow [0, 1]$, R is the scalar reward received by the agent after moving from one state to another given it took a particular action i.e. $R : S \times A \times S \rightarrow \mathbb{R}$, and $\gamma \in [0, 1]$ is the discount factor indicating how much the agent values future rewards.

Reinforcement learning (RL) (Sutton & Barto, 1998) is a type of machine learning that builds upon behavioral psychology, where a learner aims to learn from experience by sequentially making decisions in some environment. More specifically, it involves a learning agent transitioning from one state to another after taking some action in an environment, and typically receiving some reward for its transition and action choice. Ultimately, the agent seeks to learn a policy that maps states to actions that will maximize its (discounted) cumulative reward i.e. it aims to solve $\max_{\pi : S \rightarrow A} \sum_{i=0}^{\infty} \gamma^t R_t$ to find a policy π where S is the state space of the learner, A is the action space of the learner, γ is the reward discount factor, and R_t is the reward received at time-step t by the agent after taking action a_t when in state s_t.

3.2. Imitation Learning (IL)

Learning solely from experience can be very expensive. It can sometimes be intractible for an agent to fully explore the state space to converge to an optimal policy. This is especially the case in real-world robotics, where exploration must be done in real time and can incur large costs due to safety considerations. A popular solution to alleviate this problem is for some expert to guide the learner to the optimal policy through imitation learning (IL).

Conventional IL involves showing a learner an expert demonstration in the form of state-action pairs, $D^e = \{(s_t^e, a_t^e)\}$ where s_t^e is the state of the expert and a_t^e is the action taken by the expert at time t, and the goal is to learn a policy π that correctly produces a behavior similar to the expert demonstration.

A major limitation of conventional IL is that the learner needs access to the expert actions, $\{a_t^e\}$. This assumption is not necessarily practical, since many demonstrations do not have expert actions, and collecting this data can be expensive. Moreover, there are a large number of online demonstration videos that do not contain any expert information; it would be tremendously beneficial if we can exploit this valuable data without dependence on expert action information.

IL in the absence of expert action information is called imitation from observation (IfO). That is, we are trying to learn the same policy mapping π but we do not have access to expert actions. Here, our expert demonstration is of the form $D^e = \{s_t^e\}$ i.e. the learner is shown only the states of the expert. In this case, we cannot simply apply behavioral cloning since we do not have any labels; instead, we must infer the expert actions $\{a_t^e\}$ to get $\{\widetilde{a_t^e}\}$ for each state $\{s_t^e\}$ to retrieve $\widetilde{D^e} = \{(s_t^e, \widetilde{a_t^e})\}$. In this work, we focus on building an IfO control algorithm.

4. Reinforced Inverse Dynamics Modeling

We consider the problem of inferring an expert's actions, $\{a_t^e\}$, given a single state-only expert demonstration, $D^e = \{s_t^e\}$, where each s_t^e is the raw state of the expert per time-step. We propose RIDM, a method of integrating reinforcement learning and imitation from observation to learn an inverse dynamics model to perform imitation from a single expert demonstration using only raw states. In this framework, our inverse dynamics model, paramaterized by θ, maps state-transitions from the imitator's current state, s_t, to the desired expert's state, s_{t+1}^e, at time-step t, to an inferred action $\widetilde{a_t^e}$. Our inverse dynamics model learns this mapping appropriately such that $\{\widetilde{a_t^e}\}$ maximizes the cumulative reward from the environment, R_{env}.

We now provide an overall sketch of RIDM. The algorithm first randomly initializes θ, the parameters of our inverse dynamics model. Then if a known exploration policy, π^{pre}, is available, the algorithm collects π^{pre}'s state-action pairs $\{(s_t^{pre}, a_t^{pre})\}$, else we use the randomly-initialized parameters. The algorithm then applies the inverse dynamics model on the state-transition pairs of the agent's current state to the desired pre-known policy state, $\{(s'_t, s^{pre}_{t+1})\}$, to get the inferred exploration policy actions, $\{\widetilde{a^{pre}_t}\}$. θ is then optimized such that the distance between the inferred exploration policy actions, $\{\widetilde{a^{pre}_t}\}$, and true exploration policy actions, $\{a^{pre}_t\}$ is reduced by optimizing Equation 1. This process is repeated until convergence. The pre-trained inverse dynamics model is then applied to the state-transition pairs of the imitator's current state to the desired expert's state, $\{(s_t, s^e_{t+1})\}$, to get the inferred expert actions, $\{\widetilde{a^e_t}\}$. The learning agent then executes $\{\widetilde{a^e_t}\}$ in the environment, and its observed states $\{s_t\}$ and cumulative environment reward, R_{env}, are collected. Finally, θ is optimized such that the R_{env} is maximized according to Equation 2. This process repeats until convergence.

In our work, we use CMA-ES to learn the parameters, θ, of our inverse dynamics model. The psuedo-code for RIDM is given in Algorithm 1.

Algorithm 1 RIDM

```
 1: Let $D^e = \{s^e_t\}$ be a single state-only demonstration of
    raw states per time-step
 2: Let $\theta$ be the parameters of the inverse dynamics model
 3: Randomly initialize $\theta$
 4: **if** $\pi^{pre}$ available **then**
 5:     Let $D^{pre} = \{(s^{pre}_t, a^{pre}_t)\}$ generated by $\pi^{pre}$
 6:     **while** not converged **do**
 7:         Infer actions, $\{\widetilde{a^{pre}_t}\}$, for $\{(s'_t, s^{pre}_{t+1})\}$ using $\theta$
 8:         Update $\theta$ by optimizing Equation 1
 9:     **end while**
10: **end if**
11: **while** not converged **do**
12:     Infer actions, $\{\widetilde{a^e_t}\}$, for $\{(s_t, s^e_{t+1})\}$ using $\theta$
13:     Execute $\{\widetilde{a^e_t}\}$
14:     Collect observed states $\{s_t\}$
15:     Collect cumulative episode reward $R_{env}$
16:     Update $\theta$ by optimizing Equation 2
17: **end while**
18: **return** $\theta$
```

4.1. Inverse Dynamics Model Pre-training

Prior to learning the inverse dynamics model parameters, θ, for imitation, we pre-train θ using a pre-known exploration policy, if available. The motivation here is that if we have access to an exploration policy with reasonable level of performance, we can use this as a starting point instead of randomly initialized values. Note that we ultimately want to infer the actions of an expert policy, whose actions are *unknown*. In this phase, we pre-train on an exploration policy, whose actions are *known*.

RIDM first initializes θ randomly from a uniform distribution. If a known exploration policy, $\pi^{pre} : s^{pre}_t \rightarrow a^{pre}_t$, is available, the algorithm executes it in the environment, and collects the transition-action pairs, $T^{pre} = \{(s^{pre}_t, a^{pre}_t, s^{pre}_{t+1})\}$. It then computes $\{\widetilde{a^{pre}_t}\}$ for each state-transition pair of the agent's current state to the desired pre-known policy state, $\{(s'_t, s^{pre}_{t+1})\}$, using θ, and then tunes θ using CMA-ES by maximizing the fitness f_1 given by Equation 1. This procedure repeats until convergence.

$$f_1 = -\frac{1}{T}\sum_{j=1}^{J}\sum_{t=1}^{T}\frac{|\widetilde{a^{pre}_{tj}} - a^{pre}_{tj}|}{\max(a^{pre}_j) - \min(a^{pre}_j)} \tag{1}$$

where T is the length of the episode, J is the size of the expert raw state we are considering, such as the number of joints angles in a robot control task at given time-step, a^{pre}_{tj} and $\widetilde{a^{pre}_{tj}}$ are the true and predicted low-level actions, such as torques in a robot control task, of the exploration policy corresponding to the jth instance in the raw state, at time step t, and a^{pre}_j are all the low-level actions for the jth instance of the low-level action across all T time steps. The benefit of Equation 1 is that it will trade-off short term errors in order to optimize the differences across a full trajectory. For a robot control task, each j can correspond to a particular joint angle and a_j can correspond to the set of torques applied to the jth joint across all T time-steps.

We normalize the absolute difference between the actions to emphasize the range of values that the true actions take on. For example, if a true value of a low-level action varies between two large values, a small absolute error may be insignificant. However, a small absolute error may be very significant if the true low-level action varies between a small range.

We use the learned θ as a seed for the inverse dynamics model reinforcement step of the algorithm detailed in Section 4.2. The pre-training step is especially useful when we have access to an exploration policy that is similar to the expert's policy even if it is suboptimal.

4.2. Inverse Dynamics Model Reinforcement

This phase is where we use the expert's demonstration as a template to behave in the environment. Depending on the availability of a reasonably performing exploration policy, the algorithm initializes θ to either the learned θ from the pre-training phase or random values. With this as a starting point, the agent learns to behave in environment as follows.

RIDM first computes $\{\widetilde{a^e_t}\}$ on the state-transition pairs of the imitator's current state to the desired expert's state, $\{(s_t, s^e_{t+1})\}$, using θ, then it executes $\{\widetilde{a^e_t}\}$ in the environment, collects the cumulative environment reward, R_{env}, and the observed states, $\{s_t\}$, and tunes θ using CMA-ES by maximizing the fitness f_2 given by Equation 2. This procedure repeats until convergence.

$$f_2 = R_{env} \tag{2}$$

where R_{env} is the cumulative reward from the environment. Unlike in the pre-training step, the expert policy is *unknown*, so we do not have access to the actions. It is important to note that while R_{env} is still used to reinforce the learning of the inverse dynamics model, the learner is guided by the expert since the inverse dynamics model is used to transition from the imitator's current state to the next *expert's* state.

5. Empirical Results

Our experiments focus on robot control tasks in the MuJoCo simulator and SimSpark robot soccer simulator. Visual results of our algorithm have been hosted online [2].

For these robot control tasks, we model our inverse dynamics model as a PID controller [3], and are interested in learning the gains of the controller. Since the PID controller accounts for the error to get from one setpoint to a desired setpoint we view the PID controller as an inverse dynamics model. We consider input and output of Equation 3 to be the raw states and low-level actions respectively. We consider the raw state and low-level action of the learner and expert to be the joint angle value for each joint and the corresponding torque applied to that joint per time-step respectively. Joint angles allow us to use the same state formulation for all considered robot control tasks. Moreover, joint angles of a robot are reasonable quantities to estimate that are not necessarily tied to task-specific information to achieve a particular behavior.

Note that for a given input state-transition and output action, the PID controller enforces a one-to-one correspondence between each joint in the raw state, s_{tj}, and its torque, a_{tj}, where the index tj represents the jth joint angle at time-step t. More concretely, for a given transition of a particular joint angle, the PID controller will output the torque required to achieve that joint angle transition.

5.1. Experimental Set-up

We conduct our experiments of our algorithm on the MuJoCo physics engine and the RoboCup SimSpark 3D simulator.

[2]https://drive.google.com/drive/folders/1y3TjeYlS2v8JH-ubnGUcxRpyGY6R8McH

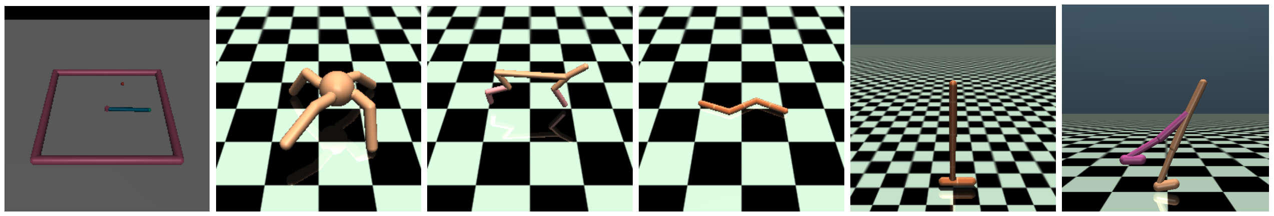
[3] Refer to the Supplementary Materials section for details about the PID controller.

5.1.1. MuJoCo Simulator

We elaborate on the specifics of each domain that we have tested on from the MuJoCo simulator (Todorov et al., 2012). In all these domains, while the state space provided by the simulator may be extensive, we make use of a very small subset of the space i.e. only the joint angles at each time-step.

- Reacher. The goal is to move a 2D robot arm to a fixed location. We use a 2 dimensional state and action space. The original state space is 11 dimensions. Since we simplify the state space to only joint angles, we fix the target location. The reward per time-step is given by the distance of the arm from the target per time-step and regularization factor of the actions.
- Ant. The goal is to make a 4-legged ant walk as fast as possible. We use an 8 dimensional state and action space. The original state space is 111 dimensions. The reward per time-step is given by the change in the global position of the ant, its forward velocity, regularization of its actions, its contact with the surface, and its survival.
- HalfCheetah. The goal is to make a cheetah walk as fast as possible. We use a 6 dimensional state and action space. The original state space is 17 dimensions. The reward per time-step is given by the cheetah's forward velocity and regularization of its actions.
- Swimmer. The goal is to make a snake-like creature swim as fast as possible in a viscous liquid. We use a 2 dimensional state and action space. The original state space is 8 dimensions. The reward per time-step is given by the swimmer's forward velocity and regularization of its actions.
- Hopper. The goal is to make a 2D one-legged robot hop as fast as possible. We use a 3 dimensional state and action space. The original state space is 11 dimensions. The reward per time-step is given by the change in the global position of the hopper, its jump height, its forward velocity, regularization of its actions, and its survival.
- Walker2d. The goal is to make a 2D bipedal robot walk as fast as possible. We use a 6 dimensional state and action space. The original state space is 17 dimensions. The reward per time-step is given by the change in the global position of the walker, its walk height, its forward velocity, regularization of its actions, and its survival.

We train the experts for each of these domains using trust region policy optimization (TRPO) (Schulman et al., 2015)



(a) Reacher (b) Ant (c) HalfCheetah (d) Swimmer (e) Hopper (f) Walker2d

Figure 1. Representative screenshots of the MuJoCo domains considered in this paper.

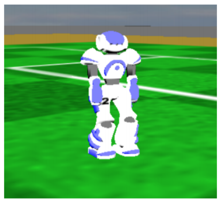


Figure 2. Simulated Nao robot in SimSpark

and proximal policy optimization (PPO) (Schulman et al., 2017), and selected those with the best performance. We use the hyperparameters specified in Schulman et al. (2015; 2017) and Henderson et al. (2017b). In our case, TRPO worked best for Reacher-v2, HalfCheetah-v2, Swimmer-v2, and Hopper-v2 and PPO worked best for Ant-v2 and Walker2d-v2.

In our experiments, we evaluate the performance of various benchmark algorithms on the full[4] and raw state spaces. Additionally, since we do not have a known exploration policy with competent performance in each of these domains, we skip the pre-training step outlined in Section 4.1.

5.1.2. SimSpark RoboCup 3D Simulation

The RoboCup 3D simulation domain is supported by two components - SimSpark (Boedecker & Asada, 2008; Xu & Vatankhah, 2014) and Open Dynamics Engine (ODE). SimSpark provides support for simulated physical multiagent system research. The ODE library enables realistic simulation of rigid body dynamics.

The agents considered in this domain are based on the Aldebaran Nao robot (Figure 2). Each agent has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck.

The goal of these experiments is to imitate certain skills of teams that participate in the yearly RoboCup competition. The challenge that arises is that these teams do not release their codebase. So we do not have access to the code or expert policies. After every yearly competition, each participating team releases their binary files, a computer readable but not human readable executable. Using these binary files we artificially create the expert demonstrations by triggering desired behaviors by, for example, placing the ball in specific locations to induce a long distance kick. In order to retrieve the state space i.e. the joint angles per time-step for specific tasks, we modify the SimSpark server to output the joint angles of the agent when performing the task.

In this domain, we use a 20 dimensional state space, where each dimension is the joint value for the respective degree of freedom and 20 dimensional action space where each dimension is the torque applied to the respective joint.

In our experiments, we are interested imitating two tasks: (1) speed walking and (2) long distance kick-offs. Since SimSpark does not have built-in reward functions, we design our own reward function. We note that the true expert may have not used our reward function.

- Speed walking. The goal of this task is to have the agent walk as fast as possible while maintaining stability throughout the episode. To do so, we define the total reward at the end of the episode to be the cumulative distance travelled per time-step with a -5 penalty for falling down. The distance is measured in meters. The reward function focuses on optimizing speed and stability.

- Long-distance kick-off. The goal of the task is to kick the ball as far as possible with highest possible elevation towards the center of the goal. To do so, we define the reward function to be

$$R_{kick} = (1 + x_{total}) \cdot \exp\left(\frac{-\theta^2}{180}\right) + x_{air} \cdot 100$$

with a -5 penalty for slightly bumping the ball, -10 penalty for falling down, where x_{total} is the distance travelled by the ball along the x-axis, θ is the angle of deviation of the ball's trajectory from the straight line between the agent and center of the goal, and x_{air} is the distance along the x-axis for which the ball was travelling in the air. x_{total} and x_{air} are in meters, and

[4] In the supplementary section, we show that this extensive state space is important for other methods to perform well on MuJoCo.

Table 1. Comparison of our method with state-of-the-art methods on the MuJoCo domain (v2) on the same *single* expert demonstration on the *raw state space (exclusively joint angles)*. Mean and standard deviations are over 100 policy runs. Performance of 0 is random and 1 is expert. *GAIL is the only method that has access to the expert actions. **Since we use a deterministic policy (fixed PID gains), we do not report mean or standard deviations of our algorithm. ⋆ Walker2d used global PID gains, unlike other domains that used local PD gains.

	Domain					
Optimization	Reacher	Ant	HalfCheetah	Swimmer	Hopper	Walker2d
GAIL*	**1.00 (0.00)**	0.08 (0.07)	0.48 (0.18)	0.34 (0.04)	0.15 (0.09)	0.03 (0.03)
GAIL* + RL	**1.00 (0.00)**	0.26 (0.04)	0.96 (0.15)	0.85 (0.04)	0.27 (0.03)	0.30 (0.10)
GAIfO	0.60 (0.09)	0.06 (0.08)	0.31 (0.19)	0.07 (0.00)	0.05 (0.02)	0.02 (0.02)
GAIfO + RL	**1.00 (0.00)**	0.23 (0.06)	0.78 (0.21)	0.44 (0.10)	0.23 (0.08)	0.19 (0.06)
BCO	-0.08 (0.16)	0.00 (0.02)	0.08 (0.04)	0.00 (0.03)	0.00 (0.01)	0.00 (0.00)
TRPO/PPO	0.99 (0.00)	0.20 (0.03)	0.37 (0.01)	0.14 (0.01)	0.18 (0.11)	0.11 (0.03)
RIDM (ours)**	**1.00**	**0.55**	**1.09**	**1.05**	**0.41**	**0.60***

θ is in degrees. The reward function values kicks that travel in the air for a long distance and exponentially decays the reward for off-target kicks.

We use two teams from the RoboCup 3D simulation league, FC Portugal (FCP) and FUT-K, as the experts, and we pre-train according to Section 4.1 on our team's walks and kicks since we have access to the actions of these exploration policies.

For SimSpark we report results using only RIDM since it is unfeasible to evaluate GAIL, GAIfO, BCO, and TRPO/PPO on the SimSpark domain due to the inefficiency of the simulator and sequential nature of these algorithms. For example, by comparing the two most time intensive tasks, we found that a single episode of the walking task in SimSpark may take up to 10 times as long as a single episode of Ant-v2.

5.2. Experimental Results

We evaluate RIDM based on the learner's ability to achieve the expert's performance. We show that by allowing the agent to maximize the cumulative environment reward and use the expert's demonstration as a template we outperform existing methods on the MuJoCo simulator and improve upon the expert's behaviors in the SimSpark simulator.

In addition to the core results shown below, we include and elaborate on additional experiments in Appendix A when trying to optimize a different fitness function given by Equation 4.

5.2.1. MuJoCo Simulator

For the MuJoCo domain we compare our imitation performance to existing baseline methods, GAIL (Ho & Ermon, 2016), GAIfO (Torabi et al., 2019b;a), and BCO (Torabi et al., 2018), using scaled performance for a particular domain where we consider 0 to be the performance achieved by a random policy and 1 to be the performance achieved by the expert policy. Note that the methods we show in Table 1 use the same settings as our algorithm i.e. a single expert demonstration with only joint angle values per time-step.

When we reduce the task-specific domain knowledge encoded in the state space to only the joint angles, we are hiding factors, such as distance from the target in a robot arm reaching task, that directly tie to high rewards. We see that the other methods when relying only on the joint angles and not optimizing for reward perform quite poorly[5].However, ours is able to significantly outperform the other methods despite using only the joint angles. We also compare our method to TRPO/PPO (Schulman et al., 2015; 2017) which focus on maximizing reward. We show that our learner achieves better performance when it is guided by the expert's demonstration instead of starting from scratch as done in the TRPO/PPO method. Finally, for a fairer comparison, we also allow GAIL and GAIfO to undergo RL by taking a linear combination of the reward outputted by their discriminators and reward from the environment. In these experiments, since the number of joints is relatively small, we use PD gains for each joint for all the domains in our experiments except Walker2d-v2, where we use global PID gains common to all joints. The maximum number parameters we are optimizing is for Ant-v2 (16).

It is important to note that GAIL is the only method that has access to the expert actions. Ours and the remaining methods do *not* have access to the expert actions. We used either TRPO or PPO depending on which one gave better performance when training the initial experts on the full state space. Refer to Table 4 in Appendix A to see performance of baseline methods using a single demonstration, but exposed to the full state space.

[5]It is shown in the supplementary materials that considering the full state space (instead of the raw states space), improves the performance of these algorithms significantly.

Table 3. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 1, and PD gains after optimizing Equation 2 (RIDM) when imitating other teams for long-distance kick offs. Performance of the experts is also presented. Note that we cannot concretely measure the reward, air distance, and angle offset achieved by the experts since we do *not* have access to their code. Hence, these are empirical estimates. We measure performance based on our reward definition, the (air) distance traveled, and angle offset. The units of distances are in meters and angles are in degrees.

Expert	Fitness	Air Distance	Distance	Angle Offset	Reward
FCP	random	0.00	0.00	0.00	-9.00
	pre-trained	0.00	2.61	1.07	-11.41
	RIDM (ours)	**13.78**	**24.05**	**3.70**	**1386.00**
	Expert	8.00	17.00	–	808.00
FUT-K	random	0.00	0.00	0.00	-9.00
	pre-trained	0.00	1.98	9.46	-13.19
	RIDM (ours)	**10.62**	**16.23**	**3.65**	**1064.00**
	Expert	0.00	10.00	–	1.00

Table 2. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 1, and PD gains after optimizing Equation 2 (RIDM) when imitating other teams for speed walking. Performance of the experts is also represented. Note that we cannot concretely measure the reward achieved by the experts and we do *not* have access to their code. Hence, these are empirical estimates. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second.

Expert	Fitness	Speed	Reward
FCP	random	0.00	-5.00
	pre-trained	0.28	3.35
	RIDM (ours)	**0.81**	**9.82**
	Expert	0.69	8.35
FUT-K	random	-0.03	-5.42
	pre-trained	0.18	2.15
	RIDM (ours)	**0.89**	**10.70**
	Expert	0.70	8.47

5.2.2. SimSpark RoboCup 3D Simulation

For each task, speed walking and long distance kick offs, we report the results for each expert team. In these experiments, since the number of joints is quite high, we use global PD gains common to all joints, so we optimize only 2 parameters.

Table 2 and Table 3 show the results of using RIDM. We report the performance achieved using randomly initialized PD gains, after pre-training, and after maximizing the cumulative environment reward (RIDM). We also report the performance of expert policies we are trying to imitate. Since we do not have access to the expert's code, we cannot concretely report the reward, the air distance, and the angle offset of these experts. Therefore, the numbers we report are empirical estimates.

We can see that by combining reinforcement learning with the expert demonstration, we are able to improve upon performance metrics relevant to the specific task compared to the expert's original performance.

6. Discussion and Future Work

In this work, we showed that our proposed algorithm, reinforced inverse dynamics modeling (RIDM), can achieve competent level of performance when a learning agent is mimicking an expert *without* access to expert actions, with only a *single* expert demonstration, and using the *raw* state space on the MuJoCo and SimSpark simulators. In particular, we showed that on the MuJoCo domain existing imitation methods heavily rely on reward information to be encoded in the state space, but our method is able to significantly outperform existing imitation techniques without this encoding in the state space. We also showed that by combining an expert demonstration with reward, we outperform methods that maximize only reward from scratch. On the SimSpark robot soccer simulator, we showed that we can develop a faster walk and longer distance kick than that of the experts. Finally, we showed a novel use-case of the PID controller as an effective inverse dynamics model.

While this lays down a framework for combining reinforcement learning and imitation from observation from a single demonstration with raw state space, there are several possible future directions. First, while CMA-ES is tremendously effective, its sample inefficiency poses a problem when applied to real robots. We would like to use a more sample efficient reinforcement learning algorithm. Second, it would also be interesting to see the performance of RIDM when using another function approximator such as a neural network as the inverse dynamics model.

Acknowledgements

We would like to thank Scott Niekum, Patrick MacAlpine, Ishan Durugkar, and the anonymous reviewer for their 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 (IIS-1637736, IIS-1651089, IIS-1724157), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Intel, Raytheon, and Lockheed Martin. 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

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009. ISSN 0921-8890. doi: 10.1016/j.robot.2008.10.024. URL http://dx.doi.org/10.1016/j.robot.2008.10.024.

Boedecker, J. and Asada, M. Simspark–concepts and application in the robocup 3d soccer simulation league. In *SIMPAR-2008 Workshop on the Universe of RoboCup Simulators*, pp. 174–181, 2008.

Depinet, M., MacAlpine, P., and Stone, P. Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the robocup 3d simulation league. In Bianchi, R. A. C., Akin, H. L., Ramamoorthy, S., and Sugiura, K. (eds.), *RoboCup-2014: Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 2015.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1577–1584. IEEE, 2018. URL https://arxiv.org/abs/1808.00928.

Hansen, N., Müller, S. D., and Koumoutsakos, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evol. Comput.*, 11(1):1–18, March 2003. ISSN 1063-6560. doi: 10.1162/106365603321828970. URL http://dx.doi.org/10.1162/106365603321828970.

Henderson, P., Chang, W., Bacon, P., Meger, D., Pineau, J., and Precup, D. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. *CoRR*, abs/1709.06683, 2017a. URL http://arxiv.org/abs/1709.06683.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017b. URL http://arxiv.org/abs/1709.06560.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017. URL http://arxiv.org/abs/1704.03732.

Ho, J. and Ermon, S. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

Hosu, I. and Rebedea, T. Playing atari games with deep reinforcement learning and human checkpoint replay. *CoRR*, abs/1607.05077, 2016. URL http://arxiv.org/abs/1607.05077.

Knox, W. B. and Stone, P. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, May 2010.

Knox, W. B. and Stone, P. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.

Lakshminarayanan, A. S., Ozair, S., and Bengio, Y. Reinforcement learning with few expert demonstrations. In *NIPS Workshop on Deep Learning for Action and Interaction*, volume 2016, 2016.

Liu, Y., Gupta, A., Abbeel, P., and Levine, S. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *CoRR*, abs/1707.03374, 2017. URL http://arxiv.org/abs/1707.03374.

MacAlpine, P., Barrett, S., Urieli, D., Vu, V., and Stone, P. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, July 2012.

Merel, J., Tassa, Y., TB, D., Srinivasan, S., Lemmon, J., Wang, Z., Wayne, G., and Heess, N. Learning human behaviors from motion capture by adversarial imitation. *CoRR*, abs/1707.02201, 2017. URL http://arxiv.org/abs/1707.02201.

Nair, A., Chen, D., Agrawal, P., Isola, P., Abbeel, P., Malik, J., and Levine, S. Combining self-supervised learning and imitation for vision-based rope manipulation. *CoRR*, abs/1703.02018, 2017a. URL http://arxiv.org/abs/1703.02018.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. *CoRR*, abs/1709.10089, 2017b. URL http://arxiv.org/abs/1709.10089.

Pomerleau, D. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation*, 1991.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017.

Schaal, S. Learning from demonstration. In Mozer, M. C., Jordan, M. I., and Petsche, T. (eds.), *Advances in Neural Information Processing Systems 9*, pp. 1040–1046. MIT Press, 1997. URL http://papers.nips.cc/paper/1224-learning-from-demonstration.pdf.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Sermanet, P., Lynch, C., Hsu, J., and Levine, S. Time-contrastive networks: Self-supervised learning from multi-view observation. *CoRR*, abs/1704.06888, 2017. URL http://arxiv.org/abs/1704.06888.

Stadie, B. C., Abbeel, P., and Sutskever, I. Third-person imitation learning. *CoRR*, abs/1703.01703, 2017. URL http://arxiv.org/abs/1703.01703.

Subramanian, K., Isbell, Jr., C. L., and Thomaz, A. L. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents &#38; Multiagent Systems*, AAMAS '16, pp. 447–456, Richland, SC, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-4239-1. URL http://dl.acm.org/citation.cfm?id=2936924.2936990.

Sutton, R. S. and Barto, A. G. Reinforcement learning i: Introduction, 1998.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.

Torabi, F., Warnell, G., and Stone, P. Adversarial imitation learning from state-only demonstrations. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2229–2231. International Foundation for Autonomous Agents and Multiagent Systems, 2019a.

Torabi, F., Warnell, G., and Stone, P. Generative adversarial imitation from observation. In *International Conference on Machine Learning Workshop on Imitation, Intent, and Interaction (I3)*, 2019b.

Torabi, F., Warnell, G., and Stone, P. Imitation learning from video by leveraging proprioception. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019c.

Torabi, F., Warnell, G., and Stone, P. Recent advances in imitation learning from observation. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2019d.

Urieli, D., MacAlpine, P., Kalyanakrishnan, S., Bentor, Y., and Stone, P. On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In Tumer, K., Yolum, P., Sonenberg, L., and Stone, P. (eds.), *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, pp. 769–776. IFAAMAS, May 2011. ISBN 978-0-9826571-5-7.

Xu, Y. and Vatankhah, H. Simspark: An open source robot simulator developed by the robocup community. In *RoboCup 2013: Robot World Cup XVII*, pp. 632–639. Springer, 2014. ISBN 978-3-662-44468-9. doi: 10.1007/978-3-662-44468-9_59.

Zhu, Y., Wang, Z., Merel, J., Rusu, A. A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., and Heess, N. Reinforcement and imitation learning for diverse visuomotor skills. *CoRR*, abs/1802.09564, 2018. URL http://arxiv.org/abs/1802.09564.

A. Supplementary Materials

Here we include details of our inverse dynamics model and additional experiments. The section includes the following: (1) Details of the inverse dynamics model, (2) Table 4 presents performance of the baseline methods when exposed to only a single expert demonstration using the full state space, and (3) Tables 5, 6, and 7 show the performance of our algorithm when trying to optimize a different fitness function f_3 given by Equation 4 instead of f_2 given by Equation 2 on the MuJoCo and SimSpark simulators.

Proportional–Integral–Derivative (PID) Controller

The PID controller is a popular control loop feedback mechanism used in control systems. Given that we are trying to adjust some variable, the PID controller will help in accurately applying the necessary correction to reach a desired setpoint. For example, if we want a robot to move its arm from 10° to 30° (desired setpoint), the PID controller will appropriately calculate the necessary torque/force to accomplish this transition. Moreover, the PID controller is also responsive; in other words, if the force applied to move from 10° to 30° is less or more than required, it will accordingly respond and adapt.

Mathematically, the PID controller is modeled as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{3}$$

where $e(t)$ is the error between the desired setpoint and current point value, K_p, K_i, and K_d are the proportionality constants for the proportional, integral, and derivative terms respectively. Intuitively, each term means the following: the proportional term signifies that if the desired setpoint is far from our current point, we should apply a larger correction to reach there, the integral term keeps track of the cumulative error of the point from the desired setpoint at each time step, this helps in applying a large correction if we have been far from the desired set point for a long time, and finally, the derivative term represents a damping factor that controls the excessive correction that may result from the proportional and integral components.

Since the PID controller accounts for the error to get from one state, s_t, to a desired setpoint, s_{t+1}, we view the PID controller as an inverse dynamics model, a mapping from state-transitions to actions i.e. $\{(s_t, s_{t+1}) \rightarrow a_t\}$, which tells us which action a_t the agent took to go from state s_t to state s_{t+1}. We consider input and output of Equation 3 to be the raw states and low-level actions respectively.

Additional Experiments

Table 4 shows the high reliance that existing methods have on task-specific domain knowledge to be encoded in the state space. However, we see that this may not necessarily be enough, since a single expert demonstration is not enough to fully imitate the expert. This is understandable for GAIfO and BCO since there is no access to the expert actions. We also see GAIL perform poorly on Ant-v2, this may be the case since the full state space of Ant-v2 is 111 dimensions, and a single expert demonstration may not be enough to imitate the expert.

Tables 5, 6, and 7 show the performance when we try to maximize the cumulative reward from the environment *and* the negative of the absolute normalized error between the learner and expert states given by f_3:

$$f_3 = \alpha R_{env} - \frac{\beta}{T} \sum_{j=1}^{J} \sum_{t=1}^{T} \frac{|s_{tj} - s^e_{tj}|}{\max(s^e_j) - \min(s^e_j)} \tag{4}$$

where R_{env} is the cumulative reward from the environment, T is the length of the episode, J is the size of the expert raw state we are considering, such as the number of joints angles in a robot control task at given time-step, α is the scalar weight of R_{env}, β is the scalar weight on the normalized absolute error between the learner and expert states, s_t is the learner's state, s^e_t is the expert's state, s^e_j are all the values for the jth instance of the raw state across all T time-steps, and the tj index corresponds to the jth joint at time-step t. We normalize the difference between states for the same reasons we mentioned for Section 4.1. Here, α and β were accordingly set so that the order of magnitudes of the two operands were similar.

We found that when $\beta \neq 0$, we either get performance similar to when $\beta = 0$ or degraded performance. This degraded performance may be so for the similar reason that methods such as GAIL, GAIFO, and BCO that rely only on the expert demonstration do not perform well on the raw state space as shown in Table 1.

Table 4. Scaled performances of the baseline imitation learning and from observation algorithms on the MuJoCo domain (v2) using the full state space and a single expert demonstration. Performance of 0 is random and 1 is expert. *GAIL is the only method that has access to the true expert actions.

	Domain					
Optimization	Reacher	Ant	HalfCheetah	Swimmer	Hopper	Walker2d
GAIL* (Ho & Ermon, 2016)	1.00 (0.00)	-0.04 (0.11)	0.90 (0.03)	0.88 (0.05)	0.95 (0.03)	0.92 (0.20)
GAIfO (Torabi et al., 2019b)	0.69 (0.05)	-0.24 (0.36)	0.58 (0.06)	0.49 (0.04)	0.92 (0.05)	0.88 (0.27)
BCO (Torabi et al., 2018)	0.86 (0.02)	0.10 (0.07)	0.58 (0.33)	0.85 (0.02)	0.02 (0.02)	0.00 (0.00)

Table 5. Scaled performances of our method on a single expert demonstration on the raw state space (exclusively joint angles) when optimizing Equation 4 with $\beta \neq 0$ using PID architectures that gave us the best results shown in Table 1. Our method uses the environment reward *and* demonstration.

Domain	Fitness	Performance
Reacher-v2	$f_3(\alpha = 0, \beta = 1)$	0.95
	$f_3(\alpha = 1, \beta = 20)$	1.00
	$f_3(\alpha = 1, \beta = 40)$	1.00
	$f_3(\alpha = 1, \beta = 60)$	1.00
Ant-v2	$f_3(\alpha = 0, \beta = 1)$	0.22
	$f_3(\alpha = 1, \beta = 250)$	0.51
	$f_3(\alpha = 1, \beta = 500)$	0.52
	$f_3(\alpha = 1, \beta = 750)$	0.50
HalfCheetah-v2	$f_3(\alpha = 0, \beta = 1)$	0.00
	$f_3(\alpha = 1, \beta = 250)$	1.08
	$f_3(\alpha = 1, \beta = 500)$	0.82
	$f_3(\alpha = 1, \beta = 750)$	1.03
Swimmer-v2	$f_3(\alpha = 0, \beta = 1)$	0.15
	$f_3(\alpha = 1, \beta = 75)$	1.03
	$f_3(\alpha = 1, \beta = 150)$	1.02
	$f_3(\alpha = 1, \beta = 225)$	1.01
Hopper-v2	$f_3(\alpha = 0, \beta = 1)$	0.05
	$f_3(\alpha = 1, \beta = 250)$	0.35
	$f_3(\alpha = 1, \beta = 500)$	0.33
	$f_3(\alpha = 1, \beta = 750)$	0.39
Walker2d-v2	$f_3(\alpha = 0, \beta = 1)$	0.01
	$f_3(\alpha = 1, \beta = 400)$	0.49
	$f_3(\alpha = 1, \beta = 600)$	0.39
	$f_3(\alpha = 1, \beta = 800)$	0.48

Table 6. Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are meter per second.

Expert	Fitness	Speed	Reward
FCP	$f_3(\alpha = 0, \beta = 1)$	0.70	8.42
	$f_3(\alpha = 1, \beta = 0.001)$	0.65	7.75
	$f_3(\alpha = 1, \beta = 0.005)$	0.76	9.21
	$f_3(\alpha = 1, \beta = 0.05)$	0.72	8.69
FUT-K	$f_3(\alpha = 0, \beta = 1)$	0.33	3.92
	$f_3(\alpha = 1, \beta = 0.5)$	0.77	9.24
	$f_3(\alpha = 1, \beta = 1)$	0.71	8.55
	$f_3(\alpha = 1, \beta = 1.5)$	0.72	8.71

Table 7. Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ when imitating other teams for long-distance kick offs. We measure performance based on our reward definition, the (air) distance travelled, and angle offset. The units of distances are meters and angles are in degrees.

Expert	Fitness	Air Distance	Distance	Angle Offset	Reward
FCP	$f_3(\alpha = 0, \beta = 1)$	0.00	2.90	12.27	-13.31
	$f_3(\alpha = 1, \beta = 300)$	12.27	20.41	4.80	1230.00
	$f_3(\alpha = 1, \beta = 600)$	7.37	21.18	3.21	742.00
	$f_3(\alpha = 1, \beta = 900)$	7.43	22.35	2.51	750.00
FUT-K	$f_3(\alpha = 0, \beta = 1)$	0.00	2.29	12.65	-13.65
	$f_3(\alpha = 1, \beta = 250)$	6.06	16.41	1.14	608.00
	$f_3(\alpha = 1, \beta = 500)$	5.94	15.69	1.60	595.00
	$f_3(\alpha = 1, \beta = 750)$	6.33	16.27	0.81	635.00