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

Brahma S. Pavse

Committee in charge:
Professor Peter Stone (Advisor)
Professor Scott Niekum
Professor Robert van de Geijn

Department of Computer Science
University of Texas at Austin
Austin, TX 78712
brahmasp@cs.utexas.edu

May 28, 2019

Acknowledgements

I would like to thank Prof. Peter Stone for taking the time and effort in advising me on this thesis. I am extremely grateful for his incredible support throughout the process. Much thanks to Prof. Scott Niekum for serving on my committee and providing useful suggestions on this work. Thanks to Prof. Robert van de Geijn for serving on my committee.

I have been extremely fortunate to have worked in the Learning Agents Research Group (LARG) and interacted with some really smart people. Faraz Torabi and I have had many insightful and fun discussions regarding this work. He was very instrumental in laying down the initial idea for this work and providing feedback on the writing of this document. Josiah Hanna was involved in many brainstorming sessions, contributed to the ideas in this work, and has reviewed earlier drafts of this document. Patrick MacAlpine first got me into the RoboCup lab, and has continued to be a source of support since the beginning. Garrett Warnell was helpful in providing suggestions on the writing of this document as well.

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 other methods on five 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.

Contents

Acknowledgements i

Abstract ii

Contents iii

List of Figures v

List of Tables vi

1 Introduction 1

2 Related Work 1

2.1 Imitation from Observation . . . 1

2.2 Integrating Reinforcement Learning and Imitation Learning . . . 2

2.3 Robot Soccer Skill Learning . . . 2

3 Preliminaries 2

3.1 Reinforcement Learning (RL) . . . 2

3.1.1 Direct Policy Search . . . 3

3.1.2 Covariance Matrix Adaptation Evolution Strategy (CMA-ES) . . . 3

3.2 Imitation Learning (IL) . . . 4

3.2.1 Conventional Imitation Learning . . . 4

3.2.2 Imitation from Observation (IfO) . . . 5

3.3 Proportional–Integral–Derivative (PID) Controller . . . 5

3.4 MuJoCo Simulator . . . 5

3.5 Robot Soccer 3D Simulation Domain . . . 6

4 Reinforced Inverse Dynamics Modeling 6

4.1 Inverse Dynamics Model Pre-training . . . 7

4.2 Inverse Dynamics Model Reinforcement . . . 8

5 Empirical Results 8

5.1 Experimental Set-up . . . 8

5.1.1 MuJoCo Simulator . . . 9

5.1.2 SimSpark RoboCup 3D Simulation . . . 9

5.2 Experimental Results . . . 10

5.2.1 MuJoCo Simulator . . . 10

5.2.2 SimSpark RoboCup 3D Simulation 11

6 Discussion and Future Work **12**

References **13**

A Supplementary Materials **16**

A.1 Baseline Methods with Full State Space on MuJoCo 16

A.2 Sample Complexities for CMA-ES 16

A.3 Reward Maximization + State Distance Minimization Fitness Function 16

A.4 PID Controller Architecture 18

A.4.1 MuJoCo Experiments 18

A.4.2 SimSpark RoboCup 3D Simulation Experiments 19

List of Figures

1 A high level depiction of RL. Here, an agent takes some action A_t in the environment given its current state S_t, and receives a reward R_t and lands in state S_{t+1}, and this interaction repeats. 3

2 A high level depiction of direct policy search. An agent acts according to some parameters, and receives a fitness score at the conclusion of the executed behavior. The optimization algorithm uses this fitness score to tune parameters to maximize this fitness. 3

3 Illustration of CMA-ES for a 2D optimization case. Each dot in an image represents a parameter candidate. Each successive generation of candidates is sampled from a distribution that moves towards the global optimum. [Image from: https://en.wikipedia.org/wiki/CMA-ES#/media/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png] 4

4 Representative screenshots of the MuJoCo domains considered in this paper. 5

5 Coordinate system of the soccer field in the 3D simulation domain. Units are in meters. [Image from: http://simspark.sourceforge.net/wiki/images/thumb/3/31/SoccerSimulation_FieldPlan.png/600px-SoccerSimulation_FieldPlan.png] . . 6

6 Simulated Nao robot in SimSpark 6

List of Tables

1 Benchmark comparisons of state-of-the-art methods on the MuJoCo domain to our method 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. *Note that 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. 11

2 Expert and pre-training agent performance values for speed walking. Note that we cannot concretely measure the reward achieved by the experts since they do not necessarily use our reward function and we do *not* have access to their code. Hence, these are empirical estimates. The units of speed are in meter per second. 11

3 Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. 12

4 Expert and pre-training agent performance metric values for long distance kick-offs. Note that we cannot concretely measure the reward, air distance, and angle offset achieved by the experts since they do not necessarily use our reward function and we do *not* have access to their code. Hence, these are empirical estimates. The units of distances are in meters and angles are in degrees. 12

5 Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. 12

6 Scaled performances of the baseline imitation learning and from observation algorithms on the MuJoCo domain 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. 16

7 Sample complexities of CMA-ES optimization algorithm for different domains on MuJoCo and SimSpark. 16

8 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. 17

9 Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ using PID architectures that gave us the best results shown in Table 3 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. . 17

10 Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ using PID architectures that gave us the best results shown in Table 5 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. 18

11 Scaled Performances when using global PD gains i.e. a single P and D gain for all joints. Performance of 0 is random and 1 is expert. 18

12 Scaled Performances when using local PID gains i.e. P, I, and D gains for each joint. Performance of 0 is random and 1 is expert. 19

13 Scaled Performances when using global PID gains i.e. a single P, I, and D gain for all joints. Performance of 0 is random and 1 is expert. 19

14 Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used local P and D gains for each joint, and optimized 40 parameters. 19

15 Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used local P, I, and D gains for each joint, and optimized 60 parameters. 19

16 Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used global P, I, and D common to all joints, and optimized 3 parameters. 20

17 Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used local P and D gains for each joint, and optimized 40 parameters. 20

18 Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used local P, I, and D gains for each joint, and optimized 60 parameters. 20

19 Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used global P, I, and D common to all joints, and optimized 3 parameters. 21

1. Introduction

Learning from experience, or reinforcement learning (Sutton and Barto, 1998), has proven to be an effective approach of instilling intelligence into an agent. However, a major limitation is that this learning process can be extremely slow and expensive. This limitation is especially true for physical robots, where robots are expensive and prone to wear and tear. 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 demonstrations of the expert. Since expert demonstration collection is often expensive, we would like to reduce our dependence on the availability of many expert 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 can 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).

We propose RIDM, a method of combining reinforcement learning (access to the environment reward) and model-based imitation from observation (IfO) 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 motivation for our algorithm is that we use the reward as a way for the learner to explore and the state-only expert demonstration 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, and are interested in learning the gains of the PD controller to infer the actions, and 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 reinforcement learning 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 Supplementary Materials in Appendix A.

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 so far in integrating reinforcement learning and imitation learning. Finally, Section 2.3 details successful efforts of using CMA-ES for robot skill learning in simulation.

2.1. Imitation from Observation

The focus of imitation from observation (IfO) 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. (2018b), Stadie et al. (2017), Sermanet et al. (2017), and Dwibedi et al. (2018).

In model-based IfO, the aim is to model the dynamics of the agent, environment, and/or both. Two types of models are (1) inverse dynamics model and (2) forward dynamics model. In this section, we elaborate on the inverse dynamics model since it is a core component of our control algorithm. For details of using a forward dynamics model for IfO, refer to Edwards et al. (2018).

An inverse dynamics model builds a mapping from state-transitions to actions ie: $\{(s_t, s_{t+1})\} \rightarrow \{a_t\}$ (Hanna and Stone, 2017). An application of this modeling was done by 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 exactly. They do this by allowing the learner to undergo self-supervision and collect $\{(s_t, a_t, s_{t+1})\}$, which is then used to train an inverse dynamics model. The learned model is then applied on the expert demonstration to infer the expert actions. Another method of this type is behavioral cloning from observation (BCO) by Torabi et al. (2018a), 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 and 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 and 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 re-starting 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 and 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 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 UT AustinVilla's 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. Section 3.2 discusses the basics of imitation learning. Section 3.3 provides insights into the PID controller, an integral component of this work. Finally, Section 3.4 and Section 3.5 describe the domains used in our experiments, the MuJoCo simulator and RoboCup SimSpark simulator respectively.

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. $T : S \times A \times S \rightarrow [0, 1]$,

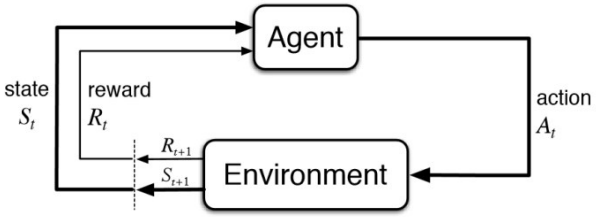


Figure 1. A high level depiction of RL. Here, an agent takes some action A_t in the environment given its current state S_t, and receives a reward R_t and lands in state S_{t+1}, and this interaction repeats.

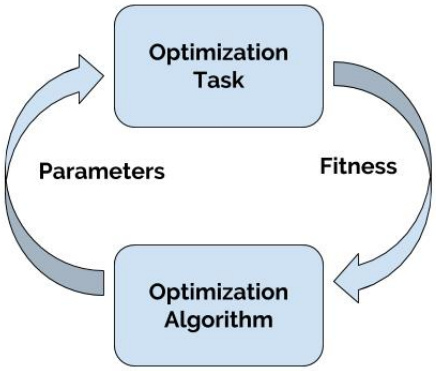


Figure 2. A high level depiction of direct policy search. An agent acts according to some parameters, and receives a fitness score at the conclusion of the executed behavior. The optimization algorithm uses this fitness score to tune parameters to maximize this fitness.

R is the scalar reward received by the agent after moving from one state to another given it took a particular action i.e. $R : S \times A \times S \to \mathbb{R}$, and $\gamma \in [0, 1]$ is the discount factor indicating how much the agent values future rewards.

Reinforcement learning (RL) (Sutton and Barto, 1998) is a type of machine learning that builds upon behavioral psychology, where a learner essentially 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 $\max_{\pi:S\to A} \sum_{i=0}^{\infty} \gamma^t R_t$ 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. Figure 1 provides an illustration of RL. Below we discuss a particular approach to RL.

3.1.1. Direct Policy Search

There are a wide range of variations to RL. A particular flavor is one that directly aims to learn parameters of a parameterized policy. During optimization, an agent executes its policy according to some parameters, and receives an overall reward or fitness score at the conclusion of the execution. The optimization algorithm then seeks to tune these parameters to maximize this overall fitness. Figure 2 provides an idea of the general flow of this optimization procedure.

3.1.2. Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

In this work, we employ CMA-ES as our direct policy search algorithm. While the details of the algorithm can be found in Hansen et al. (2003), we highlight the important high-level details here.

CMA-ES is a derivative-free stochastic optimization algorithm, which can be used to tune parameters to optimize some fitness metric. From a high-level perspective, the optimization procedure is similar to the hill-climbing strategy to maximize some function when its gradient is available i.e. the optimization moves in a direction of maximum increase of the function. Similarly, CMA-ES models a probability distribution and moves this distribution towards the region that samples parameters

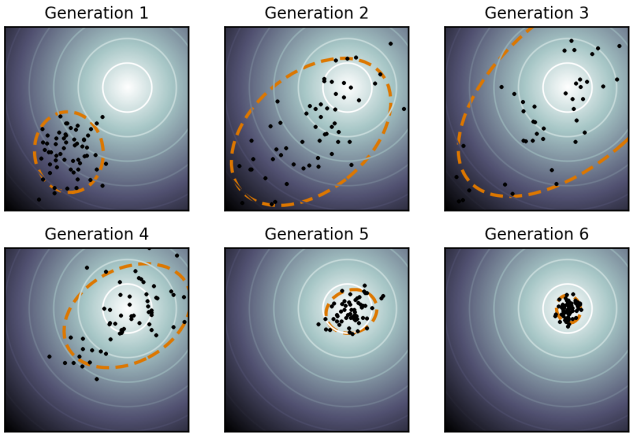


Figure 3. Illustration of CMA-ES for a 2D optimization case. Each dot in an image represents a parameter candidate. Each successive generation of candidates is sampled from a distribution that moves towards the global optimum. [Image from: https://en.wikipedia.org/wiki/CMA-ES#/media/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png]

that have high fitness scores.

Initially CMA-ES is given a seed, which specifies a multi-variate normal distribution with the initial value of the parameters to optimize along with the standard deviations for each of the parameters. At each training iteration, CMA-ES samples potential candidate parameters according to this probability distribution as a part of a population set. Each candidate is then evaluated and assigned a fitness score. CMA-ES then adjusts its model of the probability distribution to increase the probability of sampling candidates similar to those candidates that had high fitness scores in the previous iteration. Ultimately, CMA-ES aims to converge to the probability distribution that samples candidates with overall high fitness scores. Refer to Figure 3 for a pictorial representation of CMA-ES.

3.2. Imitation Learning (IL)

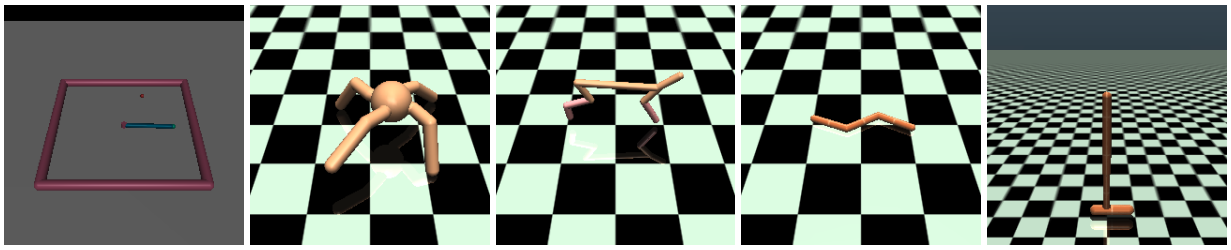
Learning solely from experience can be very expensive. It can sometimes be intractible for an agent to fully explore the state space to converge to an optimal policy. This is especially the case in real-world robotics, where exploration must be done in real time and can incur large costs due to safety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

3.2.1. Conventional Imitation Learning

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. There are two primary approaches to recovering π: (1) behavioral cloning (BC) and (2) inverse reinforcement learning (IRL). In this section, we elaborate only on BC since it provides an intuitive transition from imitation learning to imitation from observation. For a detailed understanding of IRL, refer to the work of Russell (1998) and Ng and Russell (2000).

In behavioral cloning (BC) (Bain and Sammut, 1995; Ross et al., 2010; Daftry et al., 2016), we model the problem as a supervised learning problem, and learn a policy, $\pi : s_t^e \rightarrow a_t^e$. This method has been successful on some complex tasks such as autonomous vehicle tasks (Bojarski et al., 2016; Giusti et al., 2016). While successful, a major problem with BC is the covariate shift problem (Ross and Bagnell, 2010) where, if during imitation, the learner deviates from the expert trajectory, then the error compounds for rest of the trajectory.

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.



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

Figure 4. Representative screenshots of the MuJoCo domains considered in this paper.

3.2.2. IMITATION FROM OBSERVATION (IFO)

IL in the absence of expert action information is called imitation from observation (IfO) (Liu et al., 2017). 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.

3.3. 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:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{1}$$

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 1 to be the raw states and low-level actions respectively.

3.4. MuJoCo Simulator

The MuJoCo simulator (Todorov et al., 2012) is a physics engine that is designed to help build optimization algorithms in contact-rich environments. The simulator allows us to test algorithms for optimal control and state estimations. A key feature of the simulator, which is crucial to our work, is that the inverse dynamics are accurate despite the presence of contacts such as soft contacts and dry friction.

The simulator provides a range of domains for testing. In our experiments, we have focused on Reacher-v2, Ant-v2, HalfCheetah-v2, Swimmer-v2, and Hopper-v2. Figure 4 gives a pictorial representation of each of these domains. We elaborate on the specifics of each domain in Section 5.1.1.

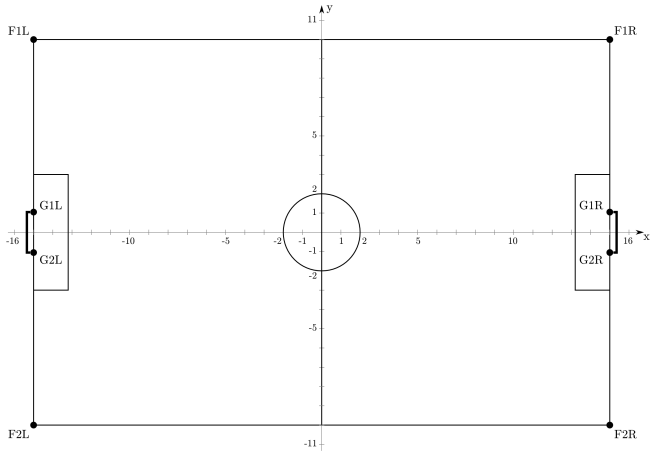


Figure 5. Coordinate system of the soccer field in the 3D simulation domain. Units are in meters. [Image from: http://simspark.sourceforge.net/wiki/images/thumb/3/31/SoccerSimulation_FieldPlan.png/600px-SoccerSimulation_FieldPlan.png]

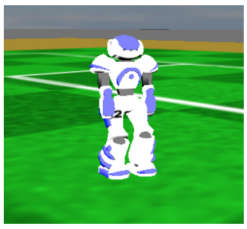


Figure 6. Simulated Nao robot in SimSpark

3.5. Robot Soccer 3D Simulation Domain

The RoboCup 3D simulation domain is supported by two components - SimSpark (Boedecker and Asada, 2008; Xu and 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. With ODE, we are also able to model hinge joints in the humanoid agents such as knee angles when an agent kicks the ball.

In RoboCup, a soccer game is between two teams, each with 11 players. The duration of a game is a total of 10 minutes (two 5 minute halves). The game takes place on a field same as the one shown in Figure 5. The figure shows the 30 X 20 sq. meter field size, location of the goals, center of the field, and bounds of the field. Each of these players (Figure 6), or agents, are based on the Aldebaran Nao robot, which has a height of about 57 cm, and a mass of 4.5 kg. Each agent has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. Each agent has the ability to determine its exact angular measurements with its perceptors and specify speed and direction when moving a joint with its effectors.

In this domain, information received and used by the agent is constrained by simulation cycles, which occur every 20 ms. Visual data, which is received every three simulation cycles, gives the distances and angles to other objects that are within the agent's vision cone (120°). Agents can also receive up to 20 byte messages from other agents every two simulation cycles.

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, $\{(s'_t, s_{t+1}^{pre})\}$, to get the inferred exploration policy actions, $\{\widetilde{a_t^{pre}}\}$. θ is then optimized such that the distance between the inferred exploration policy actions, $\{\widetilde{a_t^{pre}}\}$, and true exploration policy actions, $\{a_t^{pre}\}$ is reduced by optimizing Equation 2. 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, $\{(s_t, s_{t+1}^e)\}$, to get the inferred expert actions, $\{\widetilde{a_t^e}\}$. The learning agent then executes $\{\widetilde{a_t^e}\}$ 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 3. 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

Let $D^e = \{s_t^e\}$ be a single demonstration of raw states per time-step with no action information
Let θ be the parameters of the inverse dynamics model
Randomly initialize θ
if π^{pre} available **then**
  Let $D^{pre} = \{(s_t^{pre}, a_t^{pre})\}$ generated by π^{pre}
  while not converged **do**
    Infer actions, $\{\widetilde{a_t^{pre}}\}$, for $\{(s'_t, s_{t+1}^{pre})\}$ using θ
    Update θ by optimizing Equation 2
  end while
end if
while not converged **do**
  Infer actions, $\{\widetilde{a_t^e}\}$, for $\{(s_t, s_{t+1}^e)\}$ using θ
  Execute $\{\widetilde{a_t^e}\}$
  Collect observed states $\{s_t\}$
  Collect cumulative episode reward R_{env}
  Update θ by optimizing Equation 3
end while
return θ

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, else we use randomly-initialized parameters. 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. We now describe the pre-training process. 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, $\pi^{pre} : s_t^{pre} \rightarrow a_t^{pre}$, is available, the algorithm executes it in the environment, and collects the transition-action pairs, $T^{pre} = \{(s_t^{pre}, a_t^{pre}, s_{t+1}^{pre})\}$. It then computes $\{\widetilde{a_t^{pre}}\}$ for each state-transition pair of the agent's current state to the desired pre-known policy state, $\{(s'_t, s_{t+1}^{pre})\}$ using θ, and then tunes θ using CMA-ES by optimizing the fitness f_1 given Equation 2. This procedure repeats until convergence.

$$f_1 = -\frac{1}{T}\sum_{j=1}^{J}\sum_{t=1}^{T}\frac{|\widetilde{a_{tj}^{pre}} - a_{tj}^{pre}|}{\max(a_j^{pre}) - \min(a_j^{pre})} \tag{2}$$

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_{tj}^{pre} and $\widetilde{a_{tj}^{pre}}$ 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_j^{pre} are all the low-level actions for the jth instance of the low-level action across all T time steps. The benefit of Equation 2 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 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 $\{\widetilde{a_t^e}\}$ on the state-transition pairs of the imitator's current state to the desired expert's state, $\{(s_t, s_{t+1}^e)\}$, using θ, then it executes $\{\widetilde{a_t^e}\}$ 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 3. This procedure repeats until convergence.

$$f_2 = R_{env} \tag{3}$$

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. In this procedure, we use the reward as a way for the learner to explore and the expert's state-only demonstration as a template for ideal behavior.

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 [1].

For these robot control tasks, we model our inverse dynamics model as a PID controller, and are interested in learning the gains of the controller. 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.

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.

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

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.

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

In our experiments, we evaluate the performance of various benchmark algorithms on the full 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

We describe the relevant components of the RoboCup 3D simulation to our experiments. For a detailed description RoboCup 3D simulation as a whole refer to Section 3.5.

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 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 on MuJoCo.

5.2. Experimental Results

We evaluate our algorithm based on the learner's ability to achieve the expert's performance. We show that by allowing the agent to explore i.e. 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. The sample complexities for CMA-ES that we were used in our core results are reported in the Table 7 of the Supplementary Materials.

In addition to the core results shown below, we include additional experiments in the Supplementary Materials when trying to optimize a different fitness function given by Equation 4, which we elaborate on in that section.

5.2.1. MuJoCo Simulator

For the MuJoCo domain we compare our imitation performance to existing baseline methods, GAIL (Ho and Ermon, 2016), GAIfO (Torabi et al., 2018b), and BCO (Torabi et al., 2018a), using scaled performance for a particular domain where we consider 0 to be the performance achieved by a random agent and 1 to be the performance achieved by the expert. Note that the methods we show in Table 1 use the same settings as our algorithm i.e. a single expert demonstration with raw state space (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 methods such as GAIL (Ho and Ermon, 2016), GAIfO (Torabi et al., 2018b), and BCO (Torabi et al., 2018a) that rely only on this raw sate space i.e. joint angles and that do not optimize for reward perform quite poorly.[2] 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. 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

[2] In the supplementary materials, we show that these methods improve significantly when considering the full state space (instead of the raw state space)

Table 1. Benchmark comparisons of state-of-the-art methods on the MuJoCo domain to our method 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. *Note that 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.

	Domain				
Optimization	Reacher-v2	Ant-v2	HalfCheetah-v2	Swimmer-v2	Hopper-v2
GAIL* (Ho and Ermon, 2016)	**1.00 (0.00)**	0.08 (0.07)	0.48 (0.18)	0.34 (0.04)	0.15 (0.09)
GAIL* + RL	**1.00 (0.00)**	0.26 (0.04)	0.96 (0.15)	0.85 (0.04)	0.27 (0.03)
GAIfO (Torabi et al., 2018b)	0.60 (0.09)	0.06 (0.08)	0.31 (0.19)	0.07 (0.00)	0.05 (0.02)
GAIfO + RL	**1.00 (0.00)**	0.23 (0.06)	0.78 (0.21)	0.44 (0.10)	0.23 (0.08)
BCO (Torabi et al., 2018a)	-0.08 (0.16)	0.00 (0.02)	0.08 (0.04)	0.00 (0.03)	0.00 (0.01)
TRPO/PPO (Schulman et al., 2015; 2017)	0.99 (0.00)	0.20 (0.03)	0.37 (0.01)	0.14 (0.01)	0.18 (0.11)
RIDM (ours)**	**1.00**	**0.55**	**1.09**	**1.05**	**0.41**

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 the Supplementary Experiments for the following: (1) Table 6: performance of baseline methods using a single demonstration, but exposed to the full state space (2) Table 8: performance of our algorithm using a different fitness function given by Equation 4, which we elaborate on in that section.

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. The units are as follows: speeds are in meter per second, distances are in meters, and angles are in degrees. 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 4 shows the performance of expert policies we are trying to imitate. We also include the performance metrics and reward of our agent, which we use in the pre-training phase as outlined in Section 4.1. Since the experts do not necessarily use our reward function and that we do not have access to the code, we cannot concretely report the reward, the air distance, and the angle offset of these experts. However, the numbers we report are empirical estimates

Table 3 and Table 5 show the results of using RIDM. We report the performance achieved using randomly initialized PD gains, after pre-training, and after maximizing only the cumulative environment reward (RIDM).

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.

Refer to the Table 9 and Table 10 in the Supplementary Materials for the performance of our algorithm for speed-walking and long distance kick-offs respectively when optimizing a different fitness function given by Equation 4, which we elaborate on in that section.

Table 2. Expert and pre-training agent performance values for speed walking. Note that we cannot concretely measure the reward achieved by the experts since they do not necessarily use our reward function and we do *not* have access to their code. Hence, these are empirical estimates. The units of speed are in meter per second.

Team	Real Speed	Reward
FCP	0.69	8.35
FUT-K	0.70	8.47
Pre-known π^{pre}	0.71	8.60

Table 3. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. 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
	f_1	0.28	3.35
	RIDM (ours)	**0.81**	**9.82**
FUT-K	random	0.00	-5.00
	f_1	0.18	2.15
	RIDM (ours)	**0.89**	**10.70**

Table 4. Expert and pre-training agent performance metric values for long distance kick-offs. Note that we cannot concretely measure the reward, air distance, and angle offset achieved by the experts since they do not necessarily use our reward function and we do *not* have access to their code. Hence, these are empirical estimates. The units of distances are in meters and angles are in degrees.

Team	Air Distance	Distance	Angle Offset	Reward
FCP	8.00	17.00	—	808.00
FUT-K	0.00	10.00	—	1.00
Pre-known π^{pre}	2.09	4.82	4.11	199.00

Table 5. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 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
	f_1	0.00	2.61	1.07	-11.41
	RIDM (ours)	**13.78**	**24.05**	**3.70**	**1386.00**
FUT-K	random	0.00	0.00	0.00	-9.00
	f_1	0.00	1.98	9.46	-13.19
	RIDM (ours)	**10.62**	**16.23**	**3.65**	**1064.00**

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.

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, our method focuses on the control aspect of imitation from observation. We would like to integrate a perception component to the algorithm i.e. use raw video demonstrations to imitate the expert. Third, we would like to incorporate the imitated walking and kicking skills into our RoboCup 3D simulation team.

References

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 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.

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

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

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

Shreyansh Daftry, J. Andrew Bagnell, and Martial Hebert. Learning transferable policies for monocular reactive MAV control. *CoRR*, abs/1608.00627, 2016. URL http://arxiv.org/abs/1608.00627.

Mike Depinet, Patrick MacAlpine, and Peter Stone. Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the robocup 3d simulation league. In Reinaldo A. C. Bianchi, H. Levent Akin, Subramanian Ramamoorthy, and Komei Sugiura, editors, *RoboCup-2014: Robot Soccer World Cup XVIII*, Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 2015.

Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierre Sermanet. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1577–1584. IEEE, 2018. URL https://arxiv.org/abs/1808.00928.

Ashley D. Edwards, Himanshu Sahni, Yannick Schroecker, and Charles L. Isbell. Imitating latent policies from observation. *CoRR*, abs/1805.07914, 2018. URL http://arxiv.org/abs/1805.07914.

Alessandro Giusti, Jerome Guzzi, Dan Ciresan, Fang-Lin He, Juan Pablo Rodriguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jurgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, and Luca Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 2016.

Josiah Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017.

Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 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.

Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. *CoRR*, abs/1709.06683, 2017a. URL http://arxiv.org/abs/1709.06683.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017b. URL http://arxiv.org/abs/1709.06560.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017. URL http://arxiv.org/abs/1704.03732.

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

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

W.B̃radley Knox and Peter Stone. 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.

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

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

Yuxuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. 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.

Patrick MacAlpine, Samuel Barrett, Daniel Urieli, Victor Vu, and Peter Stone. 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.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *CoRR*, abs/1707.02201, 2017. URL http://arxiv.org/abs/1707.02201.

Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. *CoRR*, abs/1703.02018, 2017a. URL http://arxiv.org/abs/1703.02018.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *CoRR*, abs/1709.10089, 2017b. URL http://arxiv.org/abs/1709.10089.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL http://dl.acm.org/citation.cfm?id=645529.657801.

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

Stephane Ross and J. Andrew (Drew) Bagnell. Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, May 2010.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL http://arxiv.org/abs/1011.0686.

Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 101–103, New York, NY, USA, 1998. ACM. ISBN 1-58113-057-0. doi: 10.1145/279943.279964. URL http://doi.acm.org/10.1145/279943.279964.

Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017.

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

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *CoRR*, abs/1704.06888, 2017. URL http://arxiv.org/abs/1704.06888.

Bradly C. Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *CoRR*, abs/1703.01703, 2017. URL http://arxiv.org/abs/1703.01703.

Kaushik Subramanian, Charles L. Isbell, Jr., and Andrea L. Thomaz. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pages 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.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning i: Introduction, 1998.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 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*, pages 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.

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

Faraz Torabi, Garrett Warrnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018b.

Daniel Urieli, Patrick MacAlpine, Shivaram Kalyanakrishnan, Yinon Bentor, and Peter Stone. On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In Kagan Tumer, Pinar Yolum, Liz Sonenberg, and Peter Stone, editors, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, pages 769–776. IFAAMAS, May 2011. ISBN 978-0-9826571-5-7.

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

Yuke Zhu, Ziyu Wang, Josh Merel, Andrei A. Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. 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 additional experiments. The section is organized as follows. Section A.1 presents performance of the baseline methods when exposed to only a single expert demonstration using the full state space. Section A.3 discusses 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 3 on the MuJoCo and SimSpark simulators. Section A.4 includes additional experiments with various PID controller architectures.

A.1. Baseline Methods with Full State Space on MuJoCo

Table 6 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.

Table 6. Scaled performances of the baseline imitation learning and from observation algorithms on the MuJoCo domain 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-v2	Ant-v2	HalfCheetah-v2	Swimmer-v2	Hopper-v2
GAIL* (Ho and Ermon, 2016)	1.00 (0.00)	-0.04 (0.11)	0.90 (0.03)	0.88 (0.05)	0.95 (0.03)
GAIfO (Torabi et al., 2018b)	0.69 (0.05)	-0.24 (0.36)	0.58 (0.06)	0.49 (0.04)	0.92 (0.05)
BCO (Torabi et al., 2018a)	0.86 (0.02)	0.10 (0.07)	0.58 (0.33)	0.85 (0.02)	0.02 (0.02)

A.2. Sample Complexities for CMA-ES

We note the number of training iterations and populations sizes used for each domain in our core results in Section 5.2. It is important to mention that these are *not* necessarily lower-bounds since we did not optimize for sample complexity.

Table 7. Sample complexities of CMA-ES optimization algorithm for different domains on MuJoCo and SimSpark.

Domain	# Parameters	# Training Iterations	# Population Size
Reacher-v2	4	50	30
Ant-v2	16	300	150
HalfCheetah-v2	12	150	75
Swimmer-v2	4	75	30
Hopper-v2	6	250	150
SimSpark FCP (Expert 1) Walk	2	50	25
SimSpark FUT-K (Expert 2) Walk	2	50	25
SimSpark FCP (Expert 1) Kick	2	50	25
SimSpark FUT-K (Expert 2) Kick	2	50	25

A.3. Reward Maximization + State Distance Minimization Fitness Function

Tables 8, 9, and 10 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. We report results using f_3 only for PID architectures that gave us the best results in Section 5.2.

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

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 (Ho and Ermon, 2016), GAIfO (Torabi et al., 2018b), and BCO (Torabi et al., 2018a) that rely only on the expert demonstration do not perform well on the raw state space as shown in Table 1.

Table 8. 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

Table 9. Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ using PID architectures that gave us the best results shown in Table 3 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 10. Performance of our control algorithm when optimizing Equation 4 with $\beta \neq 0$ using PID architectures that gave us the best results shown in Table 5 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

A.4. PID Controller Architecture

In our experiments, since we focused on robotic control tasks, we used the PID controller as the inverse dynamics model. In our core results, certain architectures of the PID controller gave us the best result. In this section, we include additional results when trying other architectures to the PID controller.

In general, we found the following.

- Introducing the integral gain. In general, we found that it degraded performance except on Swimmer-v2. We found that the integral factor increased in value very quickly, which caused very erratic behavior due to large torques. This may have made it difficult for CMA-ES to converge to an optimal fitness. For Swimmer-v2, we were optimizing only 6 parameters, which may have been much more manageable for CMA-ES to converge.

- Global gains for all joints vs. local gains for each joint. We would except local gains per joint to give the best performance since it is less restrictive than global gains. We found that this was true for MuJoCo experiments. However, we found that global gains worked better in SimSpark. We hypothesize that introducing local gains in SimSpark creates more parameters than manageable (upto 60 parameters) for CMA-ES to optimize. In the limit, CMA-ES may be able to converge to the optimal fitness, but this was involving sample complexities much larger than those specified in Table 7. Interestingly, we found that the experiments in Table 17 achieved reasonably good results since CMA-ES was setting the integral gain close to 0.

A.4.1. MuJoCo Experiments

Table 11. Scaled Performances when using global PD gains i.e. a single P and D gain for all joints. Performance of 0 is random and 1 is expert.

Domain	Performance
Reacher-v2	1.00
Ant-v2	0.49
HalfCheetah-v2	0.22
Swimmer-v2	0.90
Hopper-v2	0.35

Table 12. Scaled Performances when using local PID gains i.e. P, I, and D gains for each joint. Performance of 0 is random and 1 is expert.

Domain ⋆	Performance
Reacher-v2	0.99
Ant-v2	0.01
HalfCheetah-v2	0.90
Swimmer-v2	1.00
Hopper-v2	0.39

Table 13. Scaled Performances when using global PID gains i.e. a single P, I, and D gain for all joints. Performance of 0 is random and 1 is expert.

Domain	Performance
Reacher-v2	1.00
Ant-v2	0.44
HalfCheetah-v2	0.46
Swimmer-v2	0.93
Hopper-v2	0.36

A.4.2. SimSpark RoboCup 3D Simulation Experiments

The results using global PD gains were reported in the core results in Table 3 and Table 5.

Table 14. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used local P and D gains for each joint, and optimized 40 parameters.

Team	Fitness	Real Speed	Reward
FCP	random	0.00	-5.00
	f_1	0.00	-5.00
	RIDM (ours)	0.00	-5.00
FUT-K	random	0.00	-5.00
	f_1	0.00	-5.00
	RIDM (ours)	0.00	-5.00

Table 15. Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used local P, I, and D gains for each joint, and optimized 60 parameters.

Team	Fitness	Real Speed	Reward
FCP	random	0.00	-5.00
	f_1	0.00	-5.00
	RIDM (ours)	0.08	-4.01
FUT-K	random	0.00	-5.00
	f_1	0.00	-5.00
	RIDM (ours)	0.06	-4.22

Table 16. Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) when imitating other teams for speed walking. We measure performance based on our reward definition and the actual speed. The units of speed are in meter per second. We used global P, I, and D common to all joints, and optimized 3 parameters.

Team	Fitness	Real Speed	Reward
FCP	random	0.00	-5.00
	f_1	0.05	-4.39
	RIDM (ours)	0.56	1.76
FUT-K	random	0.00	-5.00
	f_1	0.04	-4.55
	RIDM (ours)	0.06	-4.24

Table 17. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used local P and D gains for each joint, and optimized 40 parameters.

Expert	Fitness	Air Distance	Distance	Angle Offset	Reward
FCP	random	0.00	0.00	0.00	-9.00
	f_1	0.00	5.07	22.14	-14.60
	RIDM (ours)	0.00	4.80	19.76	-14.33
FUT-K	random	0.00	0.00	0.00	-9.00
	f_1	0.00	0.21	73.26	-15.00
	RIDM (ours)	0.00	0.23	76.12	-15.00

Table 18. Performance of randomly initialized PD gains, PD gains after pre-training i.e. optimizing Equation 2, and PD gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used local P, I, and D gains for each joint, and optimized 60 parameters.

Expert	Fitness	Air Distance	Distance	Angle Offset	Reward
FCP	random	0.00	0.93	46.38	-15.00
	f_1	0.00	1.63	33.97	-14.99
	RIDM (ours)	0.54	4.52	6.33	43.41
FUT-K	random	0.00	1.24	115.63	-15.00
	f_1	0.00	0.89	64.53	-15.00
	RIDM (ours)	0.00	0.55	4.63	-13.65

Table 19. Performance of randomly initialized PID gains, PID gains after pre-training i.e. optimizing Equation 2, and PID gains after optimizing Equation 3 (RIDM) 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 in meters and angles are in degrees. We used global P, I, and D common to all joints, and optimized 3 parameters.

Expert	Fitness	Air Distance	Distance	Angle Offset	Reward
FCP	random	0.00	0.49	76.89	-15.00
	f_1	8.31	20.26	4.06	835.39
	RIDM (ours)	10.50	23.12	4.96	1056.04
FUT-K	random	0.00	2.09	25.87	-14.92
	f_1	2.05	5.23	8.14	194.30
	RIDM (ours)	4.88	13.36	2.39	486.91