

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

Brahma S. Pavse^{†,1}, Faraz Torabi^{†,1}, Josiah Hanna², Garrett Warnell³, and Peter Stone⁴

Abstract—Augmenting reinforcement learning with imitation learning is often hailed as a method by which to improve upon learning from scratch. However, most existing methods for integrating these two techniques are subject to several strong assumptions—chief among them that information about demonstrator actions is available. In this paper, we investigate the extent to which this assumption is necessary by introducing and evaluating *reinforced inverse dynamics modeling* (RIDM), a novel paradigm for combining imitation from observation (IFO) and reinforcement learning with no dependence on demonstrator action information. Moreover, RIDM requires only a single demonstration trajectory and is able to operate directly on raw (unaugmented) state features. We find experimentally that RIDM performs favorably compared to a baseline approach for several tasks in simulation as well as for tasks on a real UR5 robot arm. Experiment videos can be found at <https://sites.google.com/view/ridm-reinforced-inverse-dynami>.

I. INTRODUCTION

Two of the most prevalent paradigms for behavior learning in artificial agents are imitation learning (IL) [1], [2] and reinforcement learning (RL) [3]. Agents that use IL receive a strong training signal in the form of an expert demonstration, but because their goal is to imitate, their task performance is typically bounded above by that of the expert. Agents using RL, on the other hand, can theoretically learn behaviors that are optimal with respect to a predefined task reward, but often have difficulty doing so due to practical challenges such as large state spaces and sparse reward functions. Because of the relative advantages and disadvantages of each of these paradigms, it is natural to investigate whether one can integrate them in order to get the best of both methods.

While combining IL and RL has been explored to a certain extent in the literature [4], [5], [6], several important issues remain. Most importantly, these techniques require access to

*This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARO (W911NF-19-2-0333), DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

[†]First Author and Second Author had equal contribution

¹First Author and Second Author are with Computer Science Department, University of Texas at Austin, USA brahmasp@utexas.edu, faraztrb@cs.utexas.edu

²Third Author is with School of Informatics, University of Edinburgh, Scotland. To be joining the Computer Sciences Department, University of Wisconsin-Madison, USA. josiah.hanna@ed.ac.uk

³Fourth Author is with Army Research Laboratory, USA garrett.a.warnell.civ@mail.mil

⁴Fifth Author is with Computer Science Department, University of Texas at Austin and Sony AI, USA. pstone@cs.utexas.edu

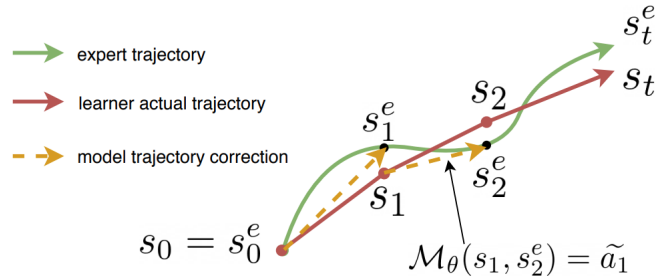


Fig. 1. RIDM applies a task-specific inverse dynamics model, \mathcal{M}_θ , on the learner's current state, s_t , to the expert's next state, s_{t+1}^e , such that the sequence of executed actions, $\{\tilde{a}_t\}$, maximizes the cumulative reward from the environment. At each time step, the agent uses the expert's next state, s_{t+1}^e (black dot), as the set point for \mathcal{M}_θ . However, it actually reaches s_{t+1} (red dot) instead—which is typically *not* the set point—since RIDM optimizes \mathcal{M}_θ to maximize environment task reward instead of minimizing trajectory-tracking error.

the internal control signals used by a demonstrator in order to be able to leverage the demonstration information [7], [6], [8]. This requirement makes it difficult to obtain useful demonstrations since it necessitates a high level of internal access to the demonstration platform, preventing, e.g., the use of numerous, easily-accessible video demonstrations available on websites like YouTube. A second limitation of many existing techniques is the requirement for *many* expert demonstrations [9], which makes obtaining sufficient demonstration data difficult in that it requires a high level of access to expert demonstrators. Finally, existing methods typically assume that they have access to *task-specific* state features during the learning process that can be used to make learning easier [7], [10], [11]. Task-specific state features are ones that somehow augment the agent's natural (or *raw* state information using additional domain knowledge—like the distance to certain important subgoals—designed to make reward function representation easier (see, e.g., Figure 2). While providing this domain knowledge may be fairly easy for a specific task, it will, in general, need to be specified anew for each new task encountered and therefore represents a practical impediment to using existing methods.

In this paper, we propose a new technique for integrating IL and RL called *reinforced inverse dynamics modeling* (RIDM) that bypasses the issues identified above. RIDM leverages recent ideas from model-based imitation from observation (IFO) to enable integrated imitation and reinforcement learning from a *single, action-free* demonstration consisting of only *raw states*. Moreover, RIDM represents a new paradigm for combining IL and RL in that the agent's behavior is based on following a fixed demonstration tra-

jectory using a parameterized task-specific inverse dynamics model (IDM) (see, Figure 1). A task-specific IDM is one that maps state-transitions to actions for a specific task only and may not generalize to other tasks that have different reward functions. While RIDM requires the demonstration trajectory during execution, its overall objective is *not* to imitate, but rather to maximize the external environment reward. RIDM accomplishes this by using RL to tune the IDM that *attempts* to follow the fixed demonstration such that the resulting behavior leads to the highest environmental reward. Formulating the overall RL problem in this way allows RIDM to diverge from the demonstration if doing so will lead to higher task reward, which is helpful when the demonstration is sub-optimal. To the best of our knowledge, we are the first to introduce an algorithm that combines IfO and RL.

To evaluate our algorithm, we establish a baseline algorithm by modifying a state-of-the-art IfO algorithm to incorporate an external reward signal. We hypothesize that RIDM will be able to outperform this baseline in the problem of interest where a few, raw-state demonstration is provided. We perform several quantitative experiments focused on both simulated and real robot control tasks, and find that RIDM’s unique, model-driven approach results in high-quality behavior trajectories that lead to better performance than the baseline.

II. RELATED WORK

This section provides a broad outline of research related to our work. The section is organized as follows. Section II-A details the most related works on imitation from observation and reinforcement learning. Section II-B discusses efforts in integrating reinforcement learning and imitation learning.

A. Imitation from Observation and Reinforcement Learning

The focus of imitation from observation (IfO) [12], [13] 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 approaches.

Many 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.[14], who show the learner a single demonstration of an expert performing some task with the intention of the learner replicating the task. Their algorithm allows the learner to undergo self-supervision by collecting states and actions, which are 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 is behavioral cloning from observation (BCO) [10], which consists of two phases. The first phase 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. The second phase involves training a policy by behavioral cloning (BC) [15], which maps the

expert states to the inferred actions. BCO, however, does not factor in the environment reward to train the inverse dynamics model or policy in either of the phases.

Iterative learning control (ILC) [16] is an older trajectory tracking approach which operates in a repetitive manner to improve its tracking precision. The methods developed in ILC, often use PID controllers and attempt to optimize the PID gains so that the agent follows the reference trajectory more accurately. Our work differs from ILC in that our objective is not accurate trajectory tracking, but rather to maximize the available environment reward, and we use RL to achieve that objective. The benefit of our work is that if the demonstration is sub-optimal, the final learned behavior could potentially outperform the demonstrator’s.

The focus of reinforcement learning is to train agent to learn a task in an environment by maximizing some notion of cumulative reward. In our work, we are focused on using black-box optimization methods. Some of the most related works are as follows. Hwangbo et al. [17] propose a method, ROCK*, for tuning a PD controller that performs favorably to CMA-ES on their experiments. Calandra et al. [18] use Bayesian Optimization to tune a state machine for robotic locomotion. They also test their method on a linear controller (which corresponds to a PD controller if the states contain positions and velocities). Neuman-Brosig et al. [19] apply Bayesian optimization for learning the parameters for active disturbance rejection control. Leonetti et al. [20] use controlled random search to tune a linear controller. Black box optimization for controller tuning has also been applied in several undergrad theses and reports [21]. Our work differs from the mentioned past work in this area in that we integrate IfO and RL which potentially helps with constraining the amount of exploration required for learning the behavior.

B. Integrating Reinforcement Learning and Imitation Learning

Another area of research related to our work is dealing with the case in which an expert demonstration 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, Taylor et al. [4] introduced Human-Agent Transfer, an algorithm that uses a human demonstration to build a base policy, which is further refined using reinforcement learning on a robot soccer domain. Lakshminarayanan et al.[5] 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 et al.[22] 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.[23] and Nair et al.[24] use IL information to alleviate the exploration process in RL. Hester et al.[8] 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. [6] optimize a

linear combination of the imitation reward outputted by generative adversarial imitation learning (GAIL) [7] and the task reward. These works assume that the learner has access to the expert’s actions.

Our work is distinct from all these works in that we focus on the integration of reinforcement learning and imitation from observation where we only have access to expert state trajectories – *not* the expert actions.

III. PRELIMINARIES

We begin by reviewing and establishing notation for reinforcement learning, imitation learning, and imitation from observation.

A. Reinforcement Learning (RL)

We model agents interacting in an environment as a Markov decision process (MDP). A MDP is denoted by the tuple $M = \langle S, A, T, R \rangle$, where S is the state space of the agent, A is the action space of the agent, T defines the environment transition function that gives the probability of the agent moving from one state to another given that the agent took a particular action (i.e., $T : S \times A \times S \rightarrow [0, 1]$), and R is the scalar-valued reward function that dictates the reward received by the agent when moving from one state to another via a particular action. In the context of the MDP framework, the reinforcement learning problem is that of optimizing the agent’s behavior so as to find a control policy, $\pi^* : S \rightarrow A$, that the agent can use to maximize the total cumulative reward it receives.

B. Imitation Learning (IL)

In contrast to reinforcement learning, imitation learning involves a learner seeking to mimic the behavior of an expert demonstrator rather than maximizing an external reward signal. We denote demonstrations as $D^e = \{(s_t^e, a_t^e)\}$, where s_t^e denotes the state of the expert at a given time index t , and a_t^e denotes the action taken by the expert at that time. Given one or many such demonstrations, the goal of IL is to learn a control policy π that the learning (imitating) agent can use to produce behavior similar to that of the expert.

IL in the absence of expert action information, i.e., when $D^e = \{s_t^e\}$, is called imitation from observation (IfO). The IfO problem is that of learning the same imitation policy π as in IL, but without access to this action information. The learner is shown only the *states* of the expert. In this case, most IL methods no longer apply, and we must find new strategies. We might try, for example, to infer the expert actions $\{a_t^e\}$ to get $\{\tilde{a}_t^e\}$ for each state $\{s_t^e\}$, and therefore approximate $\tilde{D}^e = \{(s_t^e, \tilde{a}_t^e)\}$ and then apply conventional IL methods as done by Torabi et al. [10]. In this work, we study the problem of integrating IfO with RL.

IV. REINFORCED INVERSE DYNAMICS MODELING

We now introduce reinforced inverse dynamics modeling (RIDM) – a new method for integrating IfO and RL. RIDM learns a strategy by which an agent can select actions $\{\tilde{a}_t\}$ that allow it to achieve a high level of task performance when

it has available a single, state-only expert demonstration, $D^e = \{s_t^e\}$.

RIDM does so by learning and using a task-specific inverse dynamics model (IDM), \mathcal{M}_θ , that infers which action to take at any given time instant based on both the agent’s current state and a desired next state, the *set point*. Under RIDM, the agent’s actions are computed as $\tilde{a}_t = \mathcal{M}_\theta(s_t, s_{t+1}^e)$, where s_t is the learner’s current state and s_{t+1}^e is the state of the expert at the next time instant. The goal of RIDM is to find an optimal θ such that the generated action sequence maximizes the cumulative reward from the environment, $R_{env}(\mathcal{M}_\theta)$. That is, while RIDM selects which actions to take by using the expert’s state sequence as a sequence of set points, it evaluates its policy in relation to the *environmental reward* as opposed to, e.g., the trajectory-tracking error. Note that it may actually be *desirable* for the induced state sequence to differ from that of the expert’s if doing so allows for higher environment reward. Figure 1 depicts this process. To the best of our knowledge, using such a scheme to perform integrated IfO and RL is unique in the literature.

RIDM consists of two phases. The goal of the first phase is to initialize the inverse dynamics model. This phase can either be done by selecting θ at random, or – if a known policy is available to the learner – by having the agent generate its own set of state-action-next-state triples and using supervised learning to fit θ to those triples. In the second phase, RIDM alternates between generating agent behavior according to θ and the expert demonstration, and optimizing θ in response to the amount of environment reward obtained by the generated behavior. During this phase, the learner uses the demonstration to guide the agent’s behavior (i.e., imitation), but uses the observed environment reward to adjust θ such that actions leading to high rewards are generated (i.e., reinforcement). The goal of this two-phase procedure is to find the optimal policy in terms of total task reward (which may outperform the expert) by using the expert demonstration as a guide. The pseudo-code for RIDM is given in Algorithm 1, and each phase is described in more detail below.¹

A. Inverse Dynamics Model Pre-training

During RIDM’s optional first phase, an initial value for θ is sought. This initialization is accomplished either through the use of data collected by the learner using a pre-defined exploration policy or, if such a policy is not available, by selecting the parameter value at random. We allow for RIDM to take advantage of an available exploration policy so that it can achieve a reasonable level of task performance, which is likely to get us into a good basin of attraction within the optimization landscape.

In the case where an exploration policy π^{pre} is available (e.g. if a slow-walk policy is available and we want the agent

¹Even though RIDM is described as a method that requires both environment rewards and state-only demonstrations, the algorithm can be used even if the reward is not available for instance by defining the reward as the negative of the distance between the demonstrated state and the imitator’s state at each time step.

Algorithm 1 RIDM

Require: Single, state-only demonstration $D^e := \{s_t^e\}$

- 1: **if** π^{pre} available **then**
- 2: Generate $D^{pre} := \{(s_t^{pre}, a_t^{pre})\}$ using π^{pre}
- 3: Initialize θ as the solution to (1)
- 4: **else**
- 5: Initialize θ uniformly at random
- 6: **end if**
- 7: **while** θ not converged **do**
- 8: **for** $t = 0 : |D^e| - 1$ **do**
- 9: $\tilde{a}_t := \mathcal{M}_\theta(s_t, s_{t+1}^e)$
- 10: Execute \tilde{a}_t and record s_{t+1} and reward r_t
- 11: **end for**
- 12: Compute cumulative episode reward $R_{env} = \sum_t r_t$
- 13: Update θ by solving (2)
- 14: **end while**
- 15: **return** θ^*

to learn a fast walk), RIDM computes an initial value for θ as follows. First, the learner executes π^{pre} in the environment and records the resulting experience as a trajectory of length T that we denote as $D^{pre} = \{(s_t^{pre}, a_t^{pre}, s_{t+1}^{pre})\}$. The initial value for θ is then computed by solving the following supervised learning problem:

$$\theta^* = \arg \max \left(-\frac{1}{T} \sum_{t=1}^T \sum_{n=1}^N \frac{|\mathcal{M}_\theta(s_t^{pre}, s_{t+1}^{pre})_n - a_{tn}^{pre}|}{\max(a_n^{pre}) - \min(a_n^{pre})} \right), \quad (1)$$

where N is the dimensionality of the action space, a_{tn}^{pre} denotes the scalar value of the n^{th} component of the action vector a_t^{pre} , and $\max(a_n^{pre})$ denotes the maximum value of a_{tn}^{pre} across all t . Above, notice that the goal of the optimization problem is to select θ such that $\mathcal{M}_\theta(s_t^{pre}, s_{t+1}^{pre})_n$ is a good approximation of the true action value a_{tn}^{pre} . We adopt the particular loss given above because we found that it worked well in practice. It is able to effectively trade off short-term errors in order to optimize the differences across a full trajectory, and the normalization term ensures greater accuracy for actions which vary over a smaller range. RIDM solves (1) using a blackbox optimization technique (e.g., CMA-ES[25]). Note, however, that this pre-training phase is optional, and only possible when RIDM has access to an exploration policy that generates a behavior that is qualitatively similar to the desired end behavior.

B. Inverse Dynamics Model Reinforcement

RIDM’s required second phase seeks to iteratively update the inverse dynamics model parameters in response to the environment return. The process executed here is illustrated in Figure 1, where one can see that RIDM uses the expert’s demonstration as a behavior template in the sense that the expert’s state trajectory is used as a sequence of set points to guide behavior.

The iterative updates to θ are computed as follows. First, the learner uses \mathcal{M}_θ and the expert demonstration to generate

a trajectory of experience. It does so by, when in state s_t at time step t , executing action $\tilde{a}_t = \mathcal{M}_\theta(s_t, s_{t+1}^e)$, which results in a transition to state s_{t+1} and the observation of reward r_t . After this trajectory has been generated, the learner computes the cumulative environment reward $R_{env}(D^e ; \theta) = \sum_t r_t$, which is dependent on both the (fixed) expert demonstration data D^e and the (tunable) model parameters θ . In a given iteration, i , an update to θ is computed as the solution to:

$$\theta_i = \arg \max R_{env}(D^e ; \theta_{i-1}). \quad (2)$$

It is important to note that, here, expert’s actions are *unknown*. While $R_{env}(D^e ; \theta)$ is used to reinforce the learning of the inverse dynamics model parameters, the learner is always guided by the same, fixed, state-only expert demonstration trajectory.

For each iteration of the above procedure, RIDM solves (2) again using a blackbox optimization technique (eg., CMA-ES[25] or Bayesian optimization[26]).

V. EMPIRICAL RESULTS

We now empirically validate our hypothesis, i.e., that behaviors learned using RIDM will outperform those learned by the established baseline. We focus on the case in which only a single, state-only demonstration is available to the agent and no task-specific state augmentation can be performed. Our experiments are executed in multiple robot control domains: simulated tasks are carried out in the MuJoCo and SimSpark simulators, and several manipulation tasks are carried out on a UR5 robot arm. In the first set of experiments with the MuJoCo simulator, neural networks are used to model IDMs since such models have proven effective in these domains in the literature. A neural network IDM gets one (state,next-state) pair as input and outputs the Gaussian distribution parameters from which an action is to be sampled and executed by the agent. In the rest of the experiments, i.e., the SimSpark simulator and physical tasks, PID controllers are used to model IDMs since, again, these have proven effective in these domains in the past. PID controllers are not exactly inverse dynamics models due to their differential and integral terms. However, they retain the essential characteristic that RIDM requires, i.e., that, given a current state and desired next state (or set point), they will generate an action that attempts to reach the set point. In the training process, the next state (or set point) at each time step is fixed according to the demonstration, and RIDM uses reinforcement learning to tune the IDM parameters (either NN parameters or PID gains) such that the overall agent behavior can best maximize an external environment reward. RIDM uses CMA-ES[25] in all the simulated experiments, but uses Bayesian optimization[26] in the physical robot experiments to learn the inverse dynamics model due to its superior sample complexity.

This section is organized as follows. First, we provide experimental motivation for studying RIDM in the single, state-only demonstration and raw state space case. Next, we establish a reasonable IfO+RL baseline that can also operate

in this regime, and we validate our hypothesis by comparing RIDM to this baseline. In each of our evaluations, we scale the reported performance metrics such that a score of 0 corresponds to the behavior of a random policy, and a score of 1 corresponds to the behavior of the expert. Note that, when the expert performance is sub-optimal, because we are combining imitation learning with reinforcement learning, it is reasonable to expect that the algorithm will outperform the expert on some tasks. Finally, we conclude this section by reporting additional empirical results for applying RIDM to both simulated robot soccer skill learning and to learning to perform a behavior on a physical UR5 arm robot.

A. Experimental Setup

Prior research has established that the availability of demonstrator action information and many demonstration trajectories are critical for the success of existing imitation learning algorithms [10], [11]. While RIDM is advantageous in that it does not depend on the availability of the above information, we have also claimed that it can operate directly on raw state information that has not been augmented using extra knowledge of the task which, of course, is typically unknown or difficult to obtain. We now seek to experimentally motivate the need for overcoming this issue by showing the level of reliance on these augmented state spaces in many existing imitation and reinforcement learning algorithms.

In Figure 2, we compare the scaled performance of an imitation from observation algorithm (GAIfO[11]) and a reinforcement learning algorithm (TRPO[27]) when they are given access to an augmented state space vs. when they are exposed to only the raw state space on six tasks from the MuJoCo domain [28]². Here, the raw state space refers to the list of joint angles, and the augmented state space also includes task-specific information. For instance, in the Hopper task, the augmented state space includes the agent’s global position which is advantageous in that it is more highly correlated with the reward signal (see, e.g., [11], [29], [7], [27], [30]). In this specific task, the agent’s goal is that of controlling the limbs of a 2D, one-legged robot such that it moves forward as fast as possible. The task reward given per time step corresponds to the change in global position of the agent, and since this information appears in the augmented state information, both learning algorithms perform much better (except for GAIfO in the Ant domain, perhaps due to the very large (111 dimensional) augmented state space). This advantage can be seen in Figure 2, where the high reliance of GAIfO[11] and TRPO[27] on the augmented state space is readily apparent. Such augmentations are, in general, restrictive in that they need to be redefined for each new task.

Because we seek to remove the above restriction, we use only the joint angles as the raw state information in our experiments. Many robots are comprised of joints, and therefore a joint-only representation is reasonably task-independent. The core results of our algorithm are exclu-

²The MuJoCo experiments use all the standard settings, e.g., the reward functions, the goals, and the augmented state spaces, as defined in the MuJoCo code base.

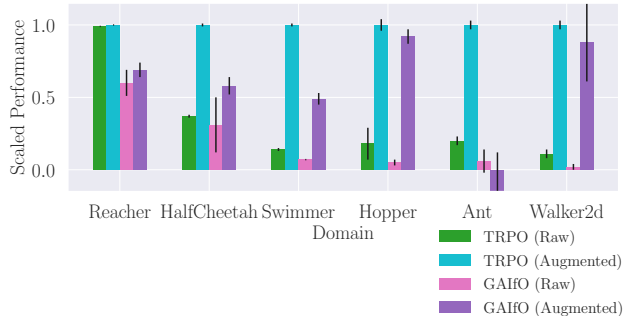


Fig. 2. Quantitative exhibition of the importance of an augmented state space for high performance on six MuJoCo domains for GAIfO and TRPO. Mean and standard deviations are over 100 policy runs. Both methods use a neural network parameterized policy.

sively concerned with dealing with the above case, i.e., single state-only demonstration consisting of joint angles.

B. RIDM Applied to MuJoCo Simulation

We now present our core results. This section is organized as follows. Section V-B.1 proposes a reasonable baseline to compare against our method. Section V-B.2 presents the performance of our algorithm against this baseline.

1) *Baseline: GAIfO+RL*: To the best of our knowledge, there is no method in the existing literature which can operate in the experimental setting of interest. Therefore, in order to understand the effectiveness of RIDM, we first propose the natural combination of the best existing algorithms for the components of RIDM, namely IfO and RL as the baseline.

GAIfO+RL is based on the current state-of-the-art imitation from observation algorithm, GAIfO[11]. Starting from GAIfO, GAIfO+RL integrates RL by modifying the reward function used during the agent update step. Instead of the reward function being determined solely by the discriminator as in GAIfO, GAIfO+RL integrates imitation and reinforcement learning by defining a new reward function that is a linear combination of the discriminator’s output and the task reward [6].

In Figure 3, we establish that GAIfO+RL is a strong baseline by evaluating the performance of GAIfO alone, RL alone (TRPO/PPO), and GAIfO+RL. All three methods operate in the raw (un-augmented) state space, and GAIfO and GAIfO+RL are also given access to a single, state-only demonstration. While the performance of either pure IfO or pure RL alone is relatively poor, we can see that GAIfO+RL achieves significantly higher performance than its parts. Moreover, GAIfO+RL operates in the same established regime and belongs to the same class of IfO+RL algorithms as RIDM, and therefore seems to be a reasonable imitation from observation + reinforcement learning algorithm.

2) *Hypothesis Validation*: We conducted an experiment comparing the scaled performance of RIDM against that of the GAIfO+RL baseline on six tasks from the MuJoCo domain. The experts are generated using TRPO/PPO with augmented states. However, the demonstrated trajectories only include the raw state information. Here, we first model

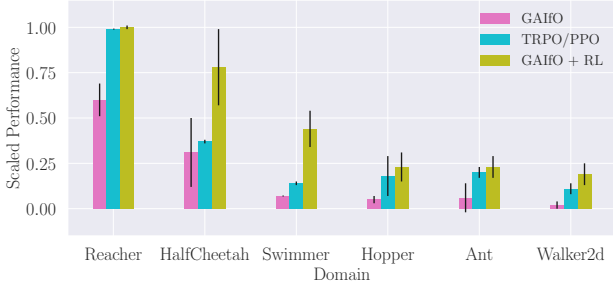


Fig. 3. Establishment of GAiFO+RL as a reasonable IfO+RL baseline to compare against RIDM. All methods use the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations are over 100 policy runs. All methods use a neural network parameterized policy.

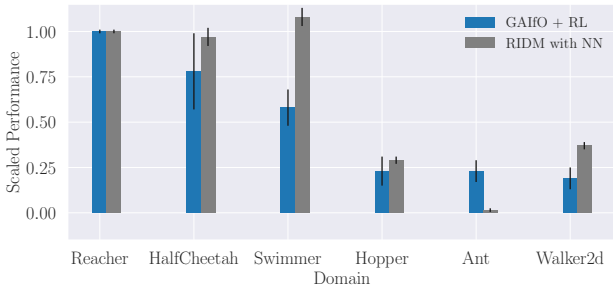


Fig. 4. Comparison of RIDM final performance against established baseline, GAiFO+RL, on the MuJoCo domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Mean and standard deviations for GAiFO+RL and RIDM are over 100 policy runs. GAiFO+RL uses a neural network parameterized policy. For each domain, in order of x -axis, the numbers of iterations required for RIDM are 700, 800, 400, 100, 900, and 1300 and for GAiFO+RL are 400, 800, 1000, 1000, 1200, and 1500

the inverse dynamics model using a neural network and train the network to maximize the received reward while attempting to follow the expert trajectory. The results are presented in Figure 4. It can be seen that RIDM outperforms GAiFO+RL in five of the domains. The only domain that the performance is worse than the baseline is the Ant domain. We speculate that the neural network IDM is not able to learn a meaningful model due to the complexity of the domain resulting from the large number of joints compared to each of the other domains. In Section VI, we show that if RIDM uses a lower-dimensional parameterized IDM (e.g. a PID controller), the performance of the learning agents is improved.

C. RIDM Applied to SimSpark RoboCup 3D Simulation

We now report the results of using RIDM to learn agent behaviors in the RoboCup 3D simulation environment, SimSpark[31], [32]. Specifically, our goal was to determine whether or not RIDM could imitate agent skills exhibited by the agents of other teams that participate in the RoboCup 3D simulation competition [33]. Since the opponent’s policies are unknown, we obtain the demonstration by executing the teams’ computer-readable but non-human-readable code in the environment.

In our experiments, we are interested in two tasks: (1) speed walking, and (2) long distance kick-offs. We collect

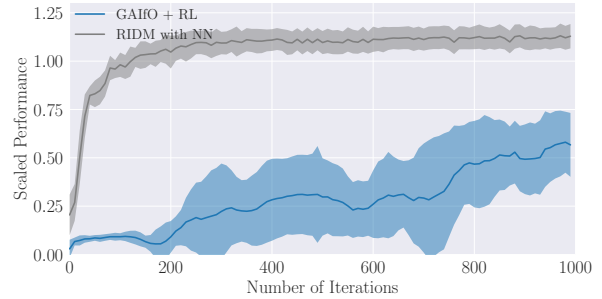


Fig. 5. Comparison of RIDM learning process against established baseline, GAiFO+RL, on the Swimmer domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Solid lines represent the mean return and shaded areas represent standard deviations over 10 trials. While the shown graph is for Swimmer-v2, we observed the same qualitative trend on other domains as well.

TABLE I
RIDM VS. EXPERT FOR SPEED WALKING.

Expert	Agent	Speed (m/s)	Reward
FCP	RIDM (ours)	0.81	9.82
	Expert	0.69	8.35
FUT-K	RIDM (ours)	0.89	10.70
	Expert	0.70	8.47

demonstration data of two teams, FC Portugal (FCP) [34] and FUT-K [35]. RIDM pre-trained the model (see Section IV-A) using walk and kick exploration policies from our own team, UT Austin Villa[33]. Here, we report results using only RIDM since it proved infeasible to evaluate GAiFO+RL in this domain due to the computational time complexity. We found that RIDM performed best with global PD gains common to all joints as the inverse dynamics model.

Below are the reward function details of each task:

- Speed walking: Summation of distances (meters) travelled per time-step with a -5 penalty for falling down.
- Long-distance kick-off:

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

with a penalty -5 for bumping the ball, -10 for falling down, where x_{total} is the x -axis distance traveled by the ball, θ is the angle between the ball’s trajectory and the line between the agent and center of the goal, and x_{air} is the x -axis distance for which the ball was traveling in the air. Distances are in meters, and θ is in degrees.

Since we defined these reward functions independently from the demonstrations, the demonstrations do not optimize the reward signals. The demonstrators are trained for the RoboCup task; their performances are sub-optimal with regards to our designed reward functions.

Tables I and II and summarize our results. We report both the performance of the expert and our agent. We can see that, since RIDM takes advantage of both the reward functions and the demonstrations, it allows our agents to *outperform* the sub-optimal experts.

TABLE II
RIDM VS. EXPERT FOR LONG-DISTANCE KICK OFFS.

Expert	Agent	x_{air} (m)	x_{total} (m)	Reward
FCP	RIDM (ours)	13.78	24.05	1386.00
	Expert	8.00	17.00	808.00
FUT-K	RIDM (ours)	10.62	16.23	1064.00
	Expert	0.00	10.00	1.00

D. RIDM Applied to a Physical UR5 Robot Arm

We also used RIDM for behavior learning on a physical robot. Specifically, we used a UR5, a 6-degree-of-freedom robotic arm. We considered a reaching task in which the arm begins in a consistent, retracted position, then must move its end effector (i.e., the gripper at the end of the arm) to a target point in Cartesian space, and finally must stop moving once the end effector has reached the target point. We trained the expert by iterating between iLQR [36] and dynamics learning with a specified reward function. We then executed this expert policy and recorded the resulting trajectories to create the demonstration data [37].

For the physical arm experiments, we skip the pre-training phase for two reasons: 1) we did not have access to a sub-optimal policy for each task, and 2) for safety concerns, we did not want to use a random exploration policy. For RIDM’s second phase, we used a reward function defined as the negative of the Euclidean distance of the end effector of the arm to the target point at each timestep. We used Bayesian optimization[26] as the blackbox optimization algorithm to update the model parameters in response to the environment reward. Bayesian optimization works by constructing a posterior distribution over the space of functions being optimized over. Here, this distribution was represented using a Gaussian process over functions that map PID values to the episode returns. As the training proceeds and more data is observed, Bayesian optimization techniques sharpen the posterior, resulting in more certainty as to which regions of the parameter space are worth exploring further with more trials and which are not. For simpler optimization problems, Bayesian optimization is more sample efficient compared to CMA-ES and converges within a few iterations.

Table III represents the results of our experiments on the UR5, where we compare RIDM to a baseline behavior generated by using the demonstration state sequence as set points for the platform’s pre-defined, hard-coded PID controller parameterization. The reported numbers are the averages and standard deviations of episode returns over five separate experiments all of which are reaching tasks with different target points. Table III shows that while RIDM outperforms the original PID, it is worse than the expert. The reason is that the expert is optimal with regards to the designed reward function.

VI. ADDITIONAL RESULTS

Due to the high performance of RIDM with PID controllers, we performed another set of experiments on the MuJoCo domains that used a PID controller as the IDM

TABLE III
RIDM VS. ORIGINAL PID CONTROLLER VS. EXPERT.

Agent	Reaching	Pushing	Pouring
RIDM (ours)	-11.94 (1.55)	-19.01 (1.03)	-5.87 (0.08)
Original PID controller	-36.57 (0.97)	-58.98 (0.15)	-15.67 (0.68)
Expert	-5.64 (0.76)	-8.43 (0.11)	-2.31 (0.04)

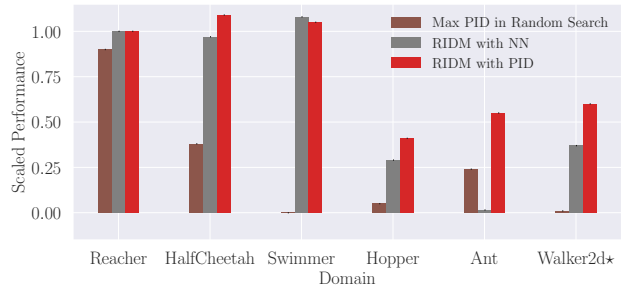


Fig. 6. Comparison of RIDM with PID as the IDM versus RIDM with NN as the IDM and the maximum performance between the randomly generated PID values on the MuJoCo domain on the same single state-only demonstration consisting of only raw states (exclusively of joint angles). Since RIDM with PID uses a deterministic inverse dynamics model, we do not report mean or standard deviations of our algorithm. *PID version of RIDM used global PID gains for Walker2d-v2, unlike on other domains where it used local PD gains.

instead of a neural network. We compare RIDM with PID controller to 1) RIDM with NN (from Figure 4) and 2) best-performing randomly generated PID gains. We determined the best-performing random PID gain by sampling 100 sets of PID gains from a Gaussian distribution with mean $[0.45, 0.75, 0.15]$ and standard deviation $[\cdot 5, \cdot 5, \cdot 5]$ (each index in the list corresponds to P, I, and D), and selecting the best-performing set. Figure 6 provides experimental results for all six of the domains. One can see that RIDM with a PID controller performs similarly to RIDM with an NN, and in more complex domains such as Ant and Walker2d, significantly outperforms it. The reasonably good performance of random PID gains shows us that even an un-trained PID controller is an effective IDM. RIDM with a PID controller is able to focus on optimizing just the (very few) parameters of the PID controller (i.e., the gains) as opposed to a neural network policy, where the policy space is much larger.

VII. CONCLUSION

In this paper, we investigated whether or not several restrictive assumptions common to many techniques that integrate imitation and reinforcement learning – access to demonstrator action information, access to several demonstrations, and knowledge of task-specific state augmentations – are necessary. We hypothesized that they are not, and we proposed a new algorithm called RIDM in order to validate that hypothesis. RIDM is a fundamentally new method for integrated imitation and reinforcement learning that operates in scenarios for which only a single, raw-state-only demonstration is provided. We experimentally demonstrated that RIDM can find behaviors that achieve good task

performance in these scenarios. Moreover, our results show that it outperforms a reasonable baseline technique while doing so. We posit that the success of RIDM is due to the way in which it generates behavior trajectories and performs learning – RIDM generates behavior by directly using the demonstration data as the set points for a parameterized but robust inverse dynamics model, and iteratively optimizes the model parameters in response to the environment reward. The above procedure not only generates reasonable trajectories over which to learn, but also reduces the learning problem to one over a relatively low-dimensional set of parameters when compared to other approaches.

This paper opens up many possible directions for future work. For one, it may be possible to extend RIDM to learn a generalized controller from numerous demonstrations of a specific task. For example, in an arm-reaching task, we may have two different demonstrations with two different target reaching points. Another open question is how RIDM will perform when using optimization algorithms other than CMA-ES, such as TRPO or PPO. Furthermore, in all of our experiments, the state spaces are low-level features (only joint angles). Another possible future direction is to investigate how RIDM performs with video demonstrations.

REFERENCES

- [1] S. Schaal, “Learning from demonstration,” in *Advances in neural information processing systems*, 1997, pp. 1040–1046.
- [2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [4] M. E. Taylor, H. B. Suay, and S. Chernova, “Integrating reinforcement learning with human demonstrations of varying ability,” in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2011, pp. 617–624.
- [5] A. S. Lakshminarayanan, S. Ozair, and Y. Bengio, “Reinforcement learning with few expert demonstrations,” in *NIPS Workshop on Deep Learning for Action and Interaction*, vol. 2016, 2016.
- [6] Y. Zhu, Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess, “Reinforcement and imitation learning for diverse visuomotor skills,” *CoRR*, vol. abs/1802.09564, 2018.
- [7] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *NIPS*, 2016, pp. 4565–4573.
- [8] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou *et al.*, “Learning from demonstrations for real world reinforcement learning,” *arXiv preprint arXiv:1704.03732*, 2017.
- [9] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
- [10] F. Torabi, G. Warnell, and P. Stone, “Behavioral cloning from observation,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.
- [11] F. Torabi, G. Warnell, and P. Stone, “Generative adversarial imitation from observation,” *arXiv preprint arXiv:1807.06158*, 2018.
- [12] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, “Imitation from observation: Learning to imitate behaviors from raw video via context translation,” *CoRR*, vol. abs/1707.03374, 2017.
- [13] F. Torabi, G. Warnell, and P. Stone, “Recent advances in imitation learning from observation,” in *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2019.
- [14] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, “Combining self-supervised learning and imitation for vision-based rope manipulation,” *CoRR*, vol. abs/1703.02018, 2017.
- [15] D. Pomerleau, “Efficient Training of Artificial Neural Networks for Autonomous Navigation,” *Neural Computation*, 1991.
- [16] D. A. Bristow, M. Tharayil, and A. G. Alleyne, “A survey of iterative learning control,” *IEEE control systems magazine*, vol. 26, no. 3, pp. 96–114, 2006.
- [17] J. Hwangbo, C. Gehring, H. Sommer, R. Siegwart, and J. Buchli, “Rockefficient black-box optimization for policy learning,” in *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 535–540.
- [18] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, “Bayesian optimization for learning gaits under uncertainty,” *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 5–23, 2016.
- [19] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, “Data-efficient autotuning with bayesian optimization: An industrial control study,” *IEEE Transactions on Control Systems Technology*, 2019.
- [20] M. Leonetti, P. Kormushev, and S. Sagratella, “Combining local and global direct derivative-free optimization for reinforcement learning,” *Cybernetics and Information Technologies*, vol. 12, no. 3, pp. 53–65, 2012.
- [21] A. Marco Valle, “Gaussian process optimization for self-tuning control,” Master’s thesis, Universitat Politècnica de Catalunya, 2015.
- [22] I. Hosu and T. Rebedea, “Playing atari games with deep reinforcement learning and human checkpoint replay,” *CoRR*, vol. abs/1607.05077, 2016.
- [23] K. Subramanian, C. L. Isbell, Jr., and A. L. Thomaz, “Exploration from demonstration for interactive reinforcement learning,” in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, ser. AAMAS ’16, 2016, pp. 447–456.
- [24] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” *CoRR*, vol. abs/1709.10089, 2017.
- [25] N. Hansen, S. D. Müller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es),” *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003.
- [26] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, “Boa: The bayesian optimization algorithm,” in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc., 1999, pp. 525–532.
- [27] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” *CoRR*, vol. abs/1502.05477, 2015.
- [28] E. Todorov, T. Erez, and Y. 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*. IEEE, 2012, pp. 5026–5033.
- [29] F. Torabi, G. Warnell, and P. Stone, “Imitation learning from video by leveraging proprioception,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.
- [30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [31] J. Bödecker and M. Asada, “Simspark concepts and application in the robocup 3 d soccer simulation league,” 2008.
- [32] Y. Xu and H. Vatankhah, “Simspark: An open source robot simulator developed by the robocup community,” in *RoboCup 2013: Robot World Cup XVII*, S. Behnke, M. Veloso, A. Visser, and R. Xiong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 632–639.
- [33] P. MacAlpine, F. Torabi, B. Pavse, J. Sigmon, and P. Stone, “UT Austin Villa: RoboCup 2018 3D simulation league champions,” in *RoboCup 2018: Robot Soccer World Cup XXII*, ser. Lecture Notes in Artificial Intelligence, D. Holz, K. Genter, M. Saad, and O. von Stryk, Eds. Springer, 2019.
- [34] L. P. Reis, N. Lau, A. Abdolmaleki, N. Shafii, R. Ferreira, A. Pereira, and D. Simões, “Fc portugal 3d simulation team: Team description paper 2017,” in *RoboCup Symposium*, 2017.
- [35] T. Iwanaga, K. Onda, and T. Yamanishi, “Fut-k team description paper 2017,”
- [36] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913.
- [37] F. Torabi, S. Geiger, G. Warnell, and P. Stone, “Sample-efficient adversarial imitation learning from observation,” *arXiv preprint arXiv:1906.07374*, 2019.

- [38] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *CoRR*, vol. abs/1709.06560, 2017.
- [39] J. Boedecker and M. Asada, "Simpark—concepts and application in the robocup 3d soccer simulation league," in *SIMPAR-2008 Workshop on the Universe of RoboCup Simulators*, 2008, pp. 174–181.
- [40] Y. Xu and H. Vatankhah, "Simpark: An open source robot simulator developed by the robocup community," in *RoboCup 2013: Robot World Cup XVII*. Springer, 2014, pp. 632–639.

VIII. SUPPLEMENTARY MATERIALS

Here we include details of our inverse dynamics model and experiment details.

A. 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:

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

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.

B. Experiment Details

1) *MuJoCo Experiments*: We train the experts for each of these domains using trust region policy optimization (TRPO) [27] and proximal policy optimization (PPO) [30], and select those with the best performance. We use the hyperparameters specified in [27] and [38]. In our case, TRPO worked best for Reacher, HalfCheetah, Swimmer, and Hopper and PPO worked best for Ant and Walker2d. Details of the considered domains are as following:

- 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.
- 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.
- 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.
- 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.

2) *3D Simulation*: The RoboCup 3D simulation domain is supported by two components - SimSpark [39], [40] and Open Dynamics Engine (ODE). SimSpark provides support for simulated physical multiagent system research. The ODE library enables realistic simulation of rigid body dynamics.

In our experiments, we are interested in 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. Refer to Appendix VIII for details about the tasks and the designed reward functions.

Since SimSpark does not have built-in reward functions, we design our own reward function. We note that the demonstrators may have not used our reward function.

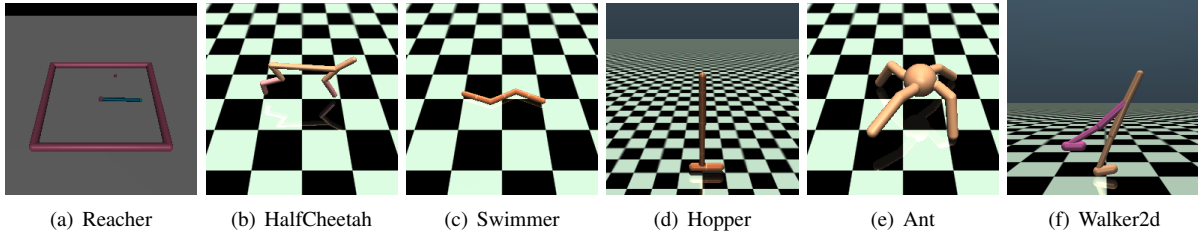


Fig. 7. Representative screenshots of the MuJoCo domains considered in this paper.

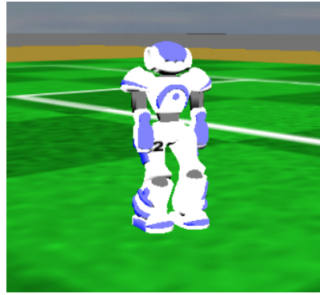


Fig. 8. Simulated Nao robot in SimSpark

- 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.
- Long-distance kick-off. The goal of the task is to kick the ball as far as possible 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.

3) *UR5 Robot Arm*: The original PID controller gains are hard-coded in the UR5 drivers when position-control mode is activated. Moreover, robots already include such a controller which is why the proposed method is so attractive, i.e., it can leverage common, pre-existing, and well-understood robotics control mechanisms.

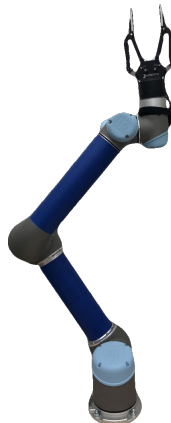


Fig. 9. UR5 Robot Arm