Receding Horizon Inverse Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Inverse reinforcement learning (IRL) seeks to infer a cost function that explains the underlying goals and preferences of expert demonstrations. This paper presents *receding horizon inverse reinforcement learning* (RHIRL), a new IRL algorithm for high-dimensional, noisy, continuous systems with black-box dynamic models. RHIRL addresses two key challenges of IRL: scalability and robustness. To handle high-dimensional continuous systems, RHIRL matches the induced optimal trajectories with expert demonstrations *locally* in a receding horizon manner and "stitches" together the local solutions to learn the cost; it thereby avoids the "curse of dimensionality". This contrasts sharply with earlier algorithms that match with expert demonstrations *globally* over the entire high-dimensional state space. To be robust against imperfect expert demonstrations and system control noise, RHIRL learns a state-dependent cost function "disentangled" from system dynamics under mild conditions. Experiments on benchmark tasks show that RHIRL outperforms several leading IRL algorithms in most instances. We also prove that the cumulative error of RHIRL grows linearly with the task duration. The code is available online[1].

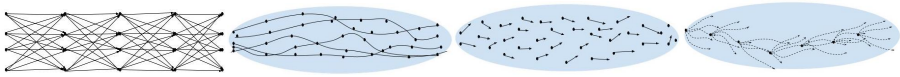
1 Introduction

Reinforcement learning (RL) has made exciting progress in a range of complex tasks, including real-time game playing [1], visuo-motor control of robots [2], and many other works. The success, however, often hinges on a carefully crafted cost function [3, 4], which is a major impediment to the wide adoption of RL in practice. Inverse reinforcement learning (IRL) [5] addresses this need by learning a cost function that explains the underlying goals and preferences of expert demonstrations. This work focuses on two key challenges in IRL, *scalability* and *robustness*.

Classic IRL algorithms commonly consist of two nested loops. The inner loop approximates the optimal control policy for a hypothesized cost function, while the outer loop updates the cost function by comparing the induced policy with expert demonstrations. The inner loop solves the (forward) reinforcement learning or optimal control problem, which is in itself a challenge for complex high-dimensional systems. Many interesting ideas have been proposed for IRL, including, e.g., maximum entropy learning [6, 7], guided cost learning [8], and adversarial learning [9]. See Figure 1 for illustrations. They try to match a *globally* optimal approximate policy with expert demonstrations over the entire system state space or a sampled approximation of it. This is impractical for high-dimensional continuous systems and is a fundamental impediment to scalability. Like RL, IRL also suffers from the "curse of dimensionality". To scale up, RHIRL computes locally optimal policies with receding horizons rather than a globally optimal policy and then matches them with expert demonstrations *locally* in succession (Figure 1*d*). The local approximation and matching substantially mitigate the impact of high-dimensional space and improve the sample efficiency of RHIRL, at the cost of a local rather than a global solution. So RHIRL trades off optimality for scalability and provides an alternative to the existing mainstream approaches.

[1] URL link removed for anonymous review.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.



(*a*) (*b*) (*c*) (*d*)

Figure 1: A comparison of RHIRL and selected IRL algorithms. They all try to match the policy induced by the learned cost with expert demonstrations. (*a*) MaxEnt matches the exact feature count over the entire system state space. (*b*) REIRL, GCL, and GAN-GCL match the approximate feature count over sampled state trajectories globally over the entire task duration. (*c*) GAIL, AIRL, ... discriminate the sampled state or state-action distributions. (*d*) RHIRL matches control sequences *locally* along demonstrated trajectories in a receding horizon manner.

Another important concern of IRL is noise in expert demonstrations and system control. Human experts may be imperfect for various reasons and provide good, but still suboptimal demonstrations. Further, the system may fail to execute the commanded actions accurately because of control noise. We want to learn a cost function that captures the expert's underlying intentions rather than the imperfectly executed actions. We recognize that there are a number of IRL works on learning from sub-optimal/fail demonstrations [10, 11, 12]. There is a subtle, but key difference between (i) demonstrations intended to be suboptimal and (ii) optimal demonstrations corrupted by noise. The existing works mentioned above address (i). RHIRL addresses (ii). To learn the true intentions from the noise corrupted demonstrations, RHIRL relies on a simplifying assumption: the cost function is linearly separable with two components, one state-dependent and one control-dependent. Many interesting systems in practice satisfy the assumption, at least, approximately [13]. RHIRL then learns the state-dependent cost component, which is disentangled from the system dynamics [9] and agnostic to noise.

2 Related Work

IRL can be viewed as an indirect approach to imitation learning. It learns a cost function, which induces an optimal policy that matches with expert demonstrations. In contrast, behavior cloning (BC) is a direct approach. It assumes independence among all demonstrated state-action pairs and learns a policy that maps states to actions through supervised learning of state-action pairs from expert demonstrations. The simplicity of BC is appealing. However, it usually requires large amounts of data to learn well and suffers from covariate shift [14]. IRL is more data-efficient. Further, it produces a cost function, which explains the expert demonstrations and potentially transfers to other systems with different dynamics. These benefits, however, come at the expense of greater computational complexity.

Classic IRL algorithms learn a cost function iteratively in a double-loop: the outer loop updates a hypothesized cost function, and the inner loop solves the forward RL problem for an optimal policy and matches it with expert demonstrations. Various methods have been proposed [6, 8, 7, 15], but they all seek a globally optimal solution over the entire state space (Figure 1*a*) [6, 7] or the entire task duration (Figure 1*b*) [8, 15]. As a result, they face significant challenges in scalability and must make simplifications, such as locally linear dynamics [8]. Recent methods use the generative adversarial network (GAN) [16] to learn a discriminator that differentiates between the state or state-action distribution induced by the learned cost and that from expert demonstrations [9, 17, 18, 19, 20, 21, 22, 23, 24]. We view this global matching as a major obstacle to scalability. In addition, GAN training is challenging and faces difficulties with convergence.

RHIRL stands in between BC and existing IRL algorithms as a novel midpoint by trading off optimality for scalability. BC performs local matching at each demonstrated state-action pair, treating all of them independently. Most of the existing IRL algorithms perform global matching over the entire state space or sampled trajectories from it. Some works [8, 25, 26] use local optimization to make the forward problem tractable, yet they [8, 25] either still still match over some global distributions, or it [26] only learns to the imitate the expert locally without any global performance guarantee. RHIRL follows the standard IRL setup. To tackle the challenge of scalability for high-dimensional continuous systems, RHIRL borrows ideas from receding horizon control [27]. It solves for locally optimal control sequences with receding horizons and learns the cost function by "stitching" together a series of locally optimal solutions to match the global state distribution of expert demonstrations (Figure 1*d*). Applying RHC to IRL has been explored in [28], yet this

work relies on a manually selected feature vector and its recursive tree-search RHC limits itself to discrete/low-dimensional tasks. A concurrent work [29] shares similar idea to learn a costmap using RHC, however it parameterizes the reward function as a linearly weighted combination of state features for simple IRL gradient derivation, hence it lacks the capacity to model other forms of reward functions.

Another challenge to IRL is suboptimal expert demonstrations and system control noise. Several methods learn an auxiliary score or ranking to reweigh the demonstrations, in order to approximate the underlying optimal expert distribution [30, 31, 22]. RHIRL does not attempt to reconstruct the optimal expert demonstrations. It explicitly models the system noise and human error in control actions as additive Gaussian noise and matches the noisy control with expert demonstrations, in order to learn from the intended, rather than the executed expert actions. Modelling the human error as additive Gaussian noise is a natural choice technically, but compromises on realism. Human errors in sequential decision making may admit other structural regularities, as a result of planning, resource constraints, uncertainty, and bounded rational pressures [32]. Studying the specific forms of human errors in the context of IRL requires insights beyond the scope of our current work and it is a promising direction for future IRL works.

3 Receding Horizon Inverse Reinforcement Learning

3.1 Overview

Consider a continuous dynamical system with noisy control:

$$x_{t+1} = f(x_t, v_t), \tag{1}$$

where $x_t \in \mathcal{R}^n$ is the state, $v_t \in \mathcal{R}^m$ is the control at time t, and the initial system state at $t = 0$ follows a distribution μ. To account for noise in expert demonstrations and in system control, we assume that v_t is a random variable following the Gaussian distribution $\mathcal{N}(v_t|u_t, \Sigma)$, with mean u_t and covariance Σ. We can control u_t directly, but not v_t, because of noise. The state-transition function f captures the system dynamics. RHIRL represents f as a black-box simulator and does not require its analytic form. Thus, we can accommodate arbitrary complex nonlinear dynamics. To simplify the presentation, we assume that the system dynamics is deterministic. We sketch the extension to stochastic dynamics at the end of the section, the full proof is given in Appendix C.

In RL, we are given a cost function and want to find a control policy that minimizes the expected total cost over time under the dynamical system. In IRL, we are not given the cost function, but instead, a set $\mathcal{D}$ of expert demonstrations. Each demonstration is a trajectory of states visited by the expert over time: $\tau = (x_0, x_1, x_2, ..., x_{T-1})$ for a duration of T steps.

We assume that the expert chooses the actions to minimize an unknown cost function and want to re this cost from the demonstrations. Formally, suppose that the cost function is parameterized by θ. RHIRL aims to learn a cost function that minimizes the loss $\mathcal{L}(\theta; \mathcal{D})$, which measures the difference between the demonstration trajectories and the optimal control policy induced by the cost function with parameters θ.

RHIRL performs this minimization iteratively, using the gradient $\partial\mathcal{L}/\partial\theta$ to update θ. In iteration t, let x_t be the system state at time t and $\mathcal{D}_t$ be the set of expert sub-trajectories starting at time t and having a duration of maximum K steps. We use the current cost to perform receding horizon control (RHC) at x_t, with time horizon K, and then update the cost by comparing the resulting state trajectory distribution with the demonstrations in $\mathcal{D}_t$. Earlier IRL work [9, 17, 15, 8, 33, 23] requires the simulator to reset to the states from the initial state distribution. To perform our receding horizon optimization, RHIRL requires a simulator to reset to the initial state x_t for the local control sequence optimization at each time step $t = 0, \ldots T-1$. This is a common setting for the work such as [34, 35, 29] from the optimal control community. See Algorithm 1 for a sketch.

First, sample M control sequences, each of length K (line 6). We assume that the covariance Σ is known. If it is unknown, we set it to be identity by default. For our experiment results reported in Table 1 and Table 2, Σ is unknown and is approximated by a constant factor of the identity matrix (grid search is performed to determine this constant factor). We show through experiments that the learned cost function is robust over different noise settings (Section 4.3). Next, we apply model-predictive path integral (MPPI) control [34] at x_t. MPPI provides an analytic solution for the

Algorithm 1 RHIRL

1: Initialize θ randomly.
2: **for** $i = 1, 2, 3, ...$ **do**
3: Sample x_0 from μ.
4: Initialize control sequence U of length K to $(0, 0, \dots)$.
5: **for** $t = 0, 1, 2, ..., T-1$ **do**
6: Sample M control sequences V_j for $j = 1, 2, \dots M$, according to $\mathcal{N}(V|U, \Sigma)$.
7: Compute the importance weight w_j for each V_j, using the state cost $S(V_j, x_t; \theta)$. See equation (11).
8: Compute $\frac{\partial}{\partial\theta}\mathcal{L}(\theta; \mathcal{D}_t, x_t)$ using $\mathcal{D}_t$ and V_j with weight w_j, for $j = 1, 2, \dots, M$. See equation (12).
9: $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$.
10: $U \leftarrow \sum_{j=1}^{M} w_j V_j$.
11: Sample v_t from $\mathcal{N}(v|u, \Sigma)$, where u is the first element in the control sequence U.
12: $x_{t+1} \leftarrow f(x_t, v_t)$.
13: Remove u from U. Append 0 at the end of U.
14: **end for**
15: **end for**

optimal control sequence distribution and the associated state sequence distribution, which allow us to estimate the gradient $\partial\mathcal{L}/\partial\theta$ efficiently through importance sampling (lines 7–8) and update cost function parameters θ (line 9). Finally, we execute the computed optimal control (line 10–12) and update the mean control input for the next iteration (line 13). We would like to emphasize that we only use the simulator to sample rollouts and evaluate the current cost function. Fixed expert demonstrations $\mathcal{D}$ are given upfront. Unlike DAgger, we do not query the expert online during training.

Our challenge is to uncover a cost function that captures the expert's intended controls $(u_0, u_1, \dots, u_{T-1})$, even though they were not directly observed, because of noise, and do so in a scalable and robust manner for high-dimensional, noisy systems.

We develop three ideas: structuring the cost function, matching locally with expert demonstrations, and efficient computation of the gradient $\partial\mathcal{L}/\partial\theta$, which are described next.

3.2 Robust Cost Functions

To learn a cost function robust against noise, we make a simplifying assumption that linearly separates the one-step cost into two components: a state cost $g(x; \theta)$ parameterized by θ and a quadratic control cost $u^T\Sigma^{-1}u$. Despite the simplification, this cost function models a wide variety of interesting systems in practice [13]. It allows RHIRL to learn a state cost $g(x; \theta)$, independent of control noise (Section 3.4), and thus generalize over different noise distributions (Section 4.3).

Suppose that $V = (v_0, v_1, \dots, v_{K-1})$ is a control sequence of length K, conditioned on the input $U = (u_0, u_1, \dots, u_{K-1})$. We apply V to the dynamical system in (1) with start state x_0 and obtain a state sequence $\tau = (x_0, x_1, \dots, x_K)$ with $x_k \sim p(x_k|x_{k-1}, v_{k-1})$ for $k = 1, 2, \dots, K$. Define the total cost of V as

$$J(V, x_0; \theta) = \sum_{k=0}^{K} g(x_k; \theta) + \sum_{k=0}^{K-1} \frac{\lambda}{2} u_k^\intercal \Sigma^{-1} u_k, \tag{2}$$

where $\lambda \geq 0$ is a constant weighting the relative importance between the state and control costs. For convenience, define also the total state cost of V as

$$S(V, x_0; \theta) = \sum_{k=0}^{K} g(x_k; \theta). \tag{3}$$

While S is defined in terms of the control sequence V, it only depends on the corresponding state trajectory τ. This is very useful, as the training data contains state and not control sequences explicitly.

3.3 Local Control Sequence Matching

To minimize the loss $\mathcal{L}$, each iteration of RHIRL applies RHC with time horizon K under the current cost parameters θ and computes locally optimal control sequences of length K. In contrast, classic

IRL algorithms, such as MaxEnt [6], perform global optimization over the entire task duration T in the inner loop. While RHC sacrifices global optimality, it is much more scalable and enables RHIRL to handle high-dimensional continuous systems. We use the hyperparameter K to trade off optimality and scalability.

Specifically, we use MPPI [34] to solve for an optimal control sequence distribution at the current start state x_t in iteration t. The main result of MPPI suggests that the optimal control sequence distribution Q minimizes the "free energy" of the dynamical system and this free energy can be calculated from the cost of the state trajectory under Q. Mathematically, the probability density $p_Q(V^*)$ can be expressed as a function of the state cost $S(V, x_t; \theta)$, with respect to a Gaussian "base" distribution $B(U_B, \Sigma)$ that depends on the control cost:

$$p_Q(V^*|U_B, \Sigma, x_t; \theta) = \frac{1}{Z}\, p_B(V^*|U_B, \Sigma) \exp\Big(-\frac{1}{\lambda} S(V^*, x_t; \theta)\Big), \tag{4}$$

where Z is the partition function. For the quadratic control cost in (2), we have $U_B = (0, 0, \ldots)$ [34]. Intuitively, the expression in (4) says that the probability density of V^* is the product of two factors, one penalizing high control cost and one penalizing high state cost. So controls with large values or resulting in high-cost states occur with probability exponentially small.

Equation (4) provides the optimal control sequence distribution under the current cost. Suppose that the control sequences for expert demonstrations $\mathcal{D}_t$ follow a distribution E. We define the loss $\mathcal{L}(\theta; \mathcal{D}_t, x_t)$ as the KL-divergence between the two distributions:

$$\mathcal{L}(\theta; \mathcal{D}_t, x_t) = D_{\text{KL}}\big(p_E(V|x_t) \parallel p_Q(V|U_B, \Sigma, x_t; \theta)\big), \tag{5}$$

which RHIRL seeks to minimize in each iteration. While the loss $\mathcal{L}$ is defined in terms of control sequence distributions, the expert demonstrations $\mathcal{D}$ provide state information and not control information. However, each control sequence V induces a corresponding state sequence τ for a given start state x_0, and τ determines the cost of V according to (2). We show in the next subsection that $\partial\mathcal{L}/\partial\theta$ can be computed efficiently using only state information from $\mathcal{D}$. This makes RHIRL appealing for learning tasks in which control labels are difficult or impossible to acquire.

3.4 Gradient Optimization

To simplify notations, we remove the explicit dependency on x_t and $\mathcal{D}_t$ and assume that in iteration t of RHIRL, all control sequences are applied with x_t as the start state and expert demonstrations come from $\mathcal{D}_t$. We have

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\theta} &= \frac{\partial}{\partial\theta}\int p_E(V) \log \frac{p_E(V)}{p_Q(V|U_B, \Sigma; \theta)} dV \\ &= \int p_E(V)\Big(\frac{1}{\lambda}\frac{\partial}{\partial\theta} S(V; \theta)\Big) dV - \int p_Q(V|U_B, \Sigma; \theta)\Big(\frac{1}{\lambda}\frac{\partial}{\partial\theta} S(V; \theta)\Big) dV\end{aligned} \tag{6}$$

The first line follows directly from the definition, and the derivation for the second line is available in Appendix A.

We estimate the two integrals in (6) through sampling. For the first integral, we can use the expert demonstrations as samples. For the second integral, we cannot sample p_Q directly, as the optimal control distribution Q is unknown in advance. Instead, we sample from a known Gaussian distribution with density $p(V|U, \Sigma)$ and apply importance sampling so that

$$\mathbb{E}_{p_Q(V|U_B, \Sigma; \theta)}[V] = \mathbb{E}_{p(V|U, \Sigma)}[w(V)V]. \tag{7}$$

The importance weight $w(V)$ is given by

$$w(V) = \frac{p_Q(V|U_B, \Sigma; \theta)}{p(V|U, \Sigma)} = \frac{p_Q(V|U_B, \Sigma; \theta)}{p_B(V|U_B, \Sigma)} \times \frac{p_B(V|U_B, \Sigma)}{p(V|U, \Sigma)} \tag{8}$$

To simplify the first ratio in (8), we substitute in the expression for p_Q from (4):

$$\frac{p_Q(V|U_B, \Sigma; \theta)}{p_B(V|U_B, \Sigma)} = \frac{1}{Z}\exp\Big(-\frac{1}{\lambda}(S(V; \theta)\Big) \tag{9}$$

We then simplify the second ratio, as both are Gaussian distributions with the same covariance matrix Σ:

$$\frac{p_B(V|U_B,\Sigma)}{p(V|U,\Sigma)} \propto \exp\left(-\sum_{k=0}^{K-1}(u_k - u_k^B)^\intercal \Sigma^{-1} v_k)\right), \tag{10}$$

where u_k and v_k, $k = 0, 1, \ldots, K-1$ are the mean controls and the sampled controls from $p(V|U,\Sigma)$, respectively. Similarly, u_k^B, $k = 0, 1, \ldots, K-1$ are the mean controls for the base distribution, and they are all 0 in our case. Substituting (9) and (10) into (8), we have

$$w(V) \propto \exp(-\frac{1}{\lambda}\left(S(V;\theta) + \lambda \sum_{k=0}^{K-1} u_k^\intercal \Sigma^{-1} v_k\right) \tag{11}$$

For each sampled control sequence V, we evaluate the expression in (11) and normalize over all samples to obtain $w(V)$.

To summarize, we estimate $\partial\mathcal{L}/\partial\theta$ through sampling:

$$\frac{\partial}{\partial\theta}\mathcal{L}(\theta;\mathcal{D}_t,x_t) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\lambda}\frac{\partial}{\partial\theta}S(V_i,x_t;\theta) - \frac{1}{M}\sum_{j=1}^{M}\frac{1}{\lambda}w(V_j)\frac{\partial}{\partial\theta}S(V_j,x_t;\theta), \tag{12}$$

where $V_i, i = 1, \ldots, N$ are the control sequences for the expert demonstrations in $\mathcal{D}_t$ and $V_j, j = 1, 2, \ldots, M$ are the sampled control sequences. Equation (12) connects $\partial\mathcal{L}/\partial\theta$ with $\partial S/\partial\theta$. The state cost function S is represented as a shallow neural network, and its derivative can be obtained easily through back-propagation. To evaluate $\frac{\partial}{\partial\theta}S(V_i,x_t;\theta)$, we do not actually use the expert control sequences, as they are unknown. We use the corresponding state trajectories in $\mathcal{D}_t$ directly, as the state cost depends only on the visited states. See equation (3).

Finally, we approximate the optimal mean control sequence according to (7):

$$U = \mathbb{E}_{p_Q(V|U_B,\Sigma,x_t;\theta)}[V] \approx \sum_{j=1}^{M} w(V_j)V_j. \tag{13}$$

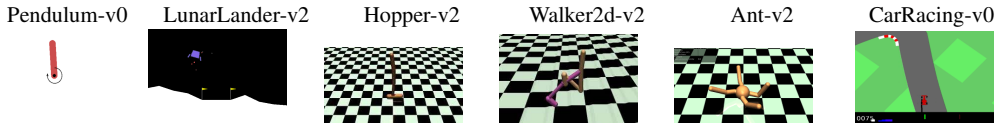
The first element in the control sequence U is the chosen control for the current time step t. We remove the first element from U and append zero at the end. The new control sequence is then used as the mean for the sampling distribution in the next iteration.

3.5 Analysis

Since RHIRL performs local optimization sequentially over many steps, one main concern is error accumulation over time. For example, standard behavior cloning has one-step error ϵ and cumulative error $O(T^2\epsilon)$ over T steps, because of covariate shift [14]. The DAgger algorithm reduces the error to $O(T\epsilon)$ by querying the expert repeatedly during online learning [14]. We prove a similar result for RHIRL, which uses offline expert demonstrations only. In iteration t of RHIRL, let $p_E(V_t|x_t)$ be the K-step expert demonstration distribution and $p_Q(V_t|U_B,\Sigma,x_t;\theta)$ be the computed K-step optimal control distribution for some fixed control parameters θ. RHIRL minimizes the KL-divergence between these two distributions in each iteration. Let $p_{\text{E}}(x)$ be the state marginal distribution of expert demonstrations and $p_{\text{RHC}}(x;\theta)$ be the state marginal distribution of the computed RHC policy under θ over the entire task duration T. Intuitively, we want the control policy under the learned cost to visit states similar to those of expert demonstrations. In other words, $p_{\text{RHC}}(x;\theta)$ and $p_{\text{E}}(x)$ are close.

Theorem 3.1. *If* $D_{\text{KL}}\big(p_E(V_t|x_t) \parallel p_Q(V_t|U_B,\Sigma,x_t;\theta)\big) < \epsilon$ *for all* $t = 0, 1, ..., T-1$, *then* $D_{\text{TV}}(p_{\text{E}}(x), p_{\text{RHC}}(x;\theta)) < T\sqrt{\epsilon/2}$.

The theorem says that RHIRL's cumulative error, measured in total variation distance D_{TV} between $p_{\text{E}}(x)$ and $p_{\text{RHC}}(x;\theta)$, grows linearly with T. The proof consists of three steps. First, in each iteration t, if the KL-divergence between two control sequence distributions are bounded by ϵ, so is the KL-divergence between the two corresponding state distributions induced by control sequences. Next, we show that the KL-divergence between the state distributions over two successive time steps are bounded by the same ϵ. Finally, we switch from KL-divergence to total variation distance and apply the triangle inequality to obtain the final result. Note that our local optimization's objective is defined



	Pendulum-v0	LunarLander-v2	Hopper-v2	Walker2d-v2	Ant-v2	CarRacing-v0
$\lvert\mathcal{S}\rvert$:	3-D	8-D	11-D	17-D	111-D	96×96 (image)
$\lvert\mathcal{A}\rvert$:	1-D	2-D	3-D	6-D	8-D	3-D
T:	100	250	1000	1000	1000	1000

Figure 2: Benchmark tasks. $|\mathcal{S}|$ and $|\mathcal{A}|$ denote the dimensions of the state space and the action space, respectively. T denotes the task horizon.

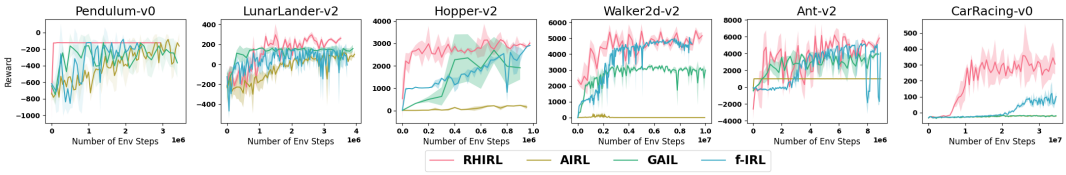


Figure 3: Learning curves for RHIRL and other methods.

in KL divergence, while the final error bound is in TV distance. We switch the distance measures to get the best of both: minimizing the KL divergence leads to strong local optimization result but KL itself is not a proper metric. Therefore, we further bound the KL divergence by TV distance to obtain a proper metric bound for the final result. The full proof is given in Appendix B. Since RHC performs local optimization in each iteration, we cannot guarantee global optimality. However, the theorem indicates that unlike standard behavior cloning, the cumulative error of RHIRL grows linearly and not quadratically in T. This shows one advantage of IRL over behavior cloning from the theoretical angle.

Given a control policy V with the resulting state marginal distribution $p_V(x)$, we can calculate the expected total cost of V by integrating the one-step cost over p_V. Now suppose that the one-step cost is bounded. Theorem 3.1 then implies that the regret in total cost, compared with the expert policy, also grows linearly in T.

3.6 Extension to Stochastic Dynamics

Suppose that the system dynamics is stochastic: $x_{t+1} = f(x_t, v_t, \omega_t)$, where ω_t is a random variable that models the independent system noise. RHIRL still applies, with modifications. We redefine the total cost functions $\tilde{J}(V, x_0; \theta)$ and $\tilde{S}(V, x_0; \theta)$ by taking expectation over the system noise distribution. When calculating the importance weight $\tilde{w}(V)$ in (11), we sample over the noise distribution to estimate the expected total state cost. Finally, we may need more samples when estimating the gradient in (12), because of the increased variance due to the system noise. The full proof is given in Appendix C. We acknowledge that the experiments in this work are of deterministic dynamics, we leave the empirical results on RHIRL's extension to stochastic dynamics to future work.

4 Experiments

We investigate two main questions. Does RHIRL scale up to high-dimensional continuous control tasks? Does RHIRL learn a robust cost function under noise?

4.1 Setup

We compare RHIRL with two leading IRL algorithms, namely AIRL [9] and f-IRL [20], and one imitation learning algorithm, GAIL [17]. In particular, f-IRL is a recent algorithm that achieves leading performance on high-dimensional control tasks. We use the implementation of AIRL, GAIL, and f-IRL from the f-IRL's official repository along with the reported hyperparameters [20], whenever possible. We also perform hyperparameter search on a grid to optimize the performance of every method on every task. The specific hyperparameter settings used are reported in Appendix D.2.

Our benchmark set consists of six continuous control tasks (Figure 2) from OpenAI Gym [36], with increasing sizes of state and action spaces. For the most complex task, CarRacing, the input consists of 96×96 raw images, resulting in an enormous state space that poses great challenges [37]. To our knowledge, RHIRL is the first few IRL algorithms to attempt such a high-dimensional space. For fair comparison, we customize all tasks to the fixed task horizon settings (Figure 2. See Appendix D.2 for details on task parameter settings.

We use WorldModel [37] to generate expert demonstration data for CarRacing and use SAC [38] for the other tasks. We add Gaussian noise to the input controls and collect expert demonstrations at different control noise levels. The covariance of the control noise is *unknown* to all methods, including, in particular, RHIRL.

To measure the performance of the learned cost function and policy, we score its induced optimal policy using the ground-truth cost function. For ease of comparison with the literature, we use negated cost values, i.e., rewards, in all reported results. Higher values indicate better performance. Each experiment is repeated 10 times to estimate the performance variance.

4.2 Scalability

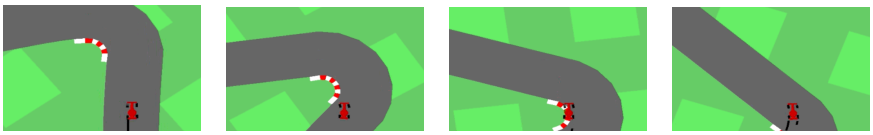
We compare RHIRL with other methods, first in noise-free environments (Figure 3) and then with increasing noise levels (Table 1).

Figure 3 shows the learning curve of each methods in noise-free environments. Overall, RHIRL converges faster and achieves higher return, especially for tasks with higher state space dimensions. This improved performance suggests that the benefit of local optimization adopted by RHIRL outweighs its potential limitations.

Table 1: Performance comparison of RHIRL and other methods. The performance is reported as the ratio of the learned policy's average return and the expert's average return. The absolute average returns and the standard deviations are reported in Appendix D.4. Negative ratios are clipped to 0. The two numbers under the name of each environment indicate the dimensions of the state space and the action space, respectively.

		No Noise $\Sigma = 0$	**Mild Noise** $\Sigma = 0.2$	**High Noise** $\Sigma = 0.5$
Pendulum 3, 1	Expert	-154.69 $\pm$ 67.61	-156.50 $\pm$ 70.72	-168.54 $\pm$ 80.89
	RHIRL	1.06	**1.07**	**1.08**
	f-IRL	**1.07**	1.06	0.93
	AIRL	1.05	0.94	0.91
	GAIL	0.88	0.89	0.80
Lunarlander 8, 2	Expert	232.00 $\pm$ 86.12	222.65 $\pm$ 56.35	164.52 $\pm$ 16.79
	RHIRL	**1.05**	**1.04**	**1.20**
	f-IRL	0.76	0.63	0.74
	AIRL	0.74	0.60	0.58
	GAIL	0.72	0.56	0.60
Hopper 11, 3	Expert	3222.48 $\pm$ 390.65	3159.72 $\pm$ 520.00	2887.72 $\pm$ 483.93
	RHIRL	0.95	**0.98**	**0.96**
	f-IRL	**0.96**	0.82	0.43
	AIRL	0.01	0.01	0.01
	GAIL	0.82	0.50	0.24
Walker2d 17, 6	Expert	4999.47 $\pm$ 55.99	4500.43 $\pm$ 114.48	3624.48 $\pm$ 95.05
	RHIRL	**0.99**	**0.99**	**0.95**
	f-IRL	**0.99**	0.82	0.78
	AIRL	0.00	0.00	0.00
	GAIL	0.50	0.64	0.51
Ant 111, 8	Expert	5759.22 $\pm$ 173.57	3257.37 $\pm$ 501.95	252.62 $\pm$ 91.44
	RHIRL	0.86	**0.93**	**0.91**
	f-IRL	**0.87**	0.80	0.78
	AIRL	0.17	0.33	0.00
	GAIL	0.48	0.40	0.00
CarRacing 96×96, 3	Expert	903.25 $\pm$ 0.23	702.01 $\pm$ 0.3	281.12 $\pm$ 0.34
	RHIRL	**0.40**	**0.29**	**0.19**
	f-IRL	0.09	0.03	0.00
	AIRL	0.00	0.00	0.00
	GAIL	0.00	0.01	0.00

Table 1 shows the final performance of all methods at various noise levels. RHIRL clearly outperforms AIRL and GAIL in all experiments. So we focus our discussion on comparison with f-IRL. In noise-free environments, RHIRL and f-IRL perform comparably on most tasks. On CarRacing, the most challenging task, RHIRL performs much better. RHIRL manages to learn the critical driving behaviors illustrated in Figure 4, despite the high-dimensional image input. However, RHIRL does not manage to learn to drive fast enough. That is the main reason why it under-performs the expert. In comparison, f-IRL only learns to follow a straight lane after a large number of environment steps, and still fails to make a sharp turn after 3.0×10^7 environment steps. In the noisy environments, the advantage of RHIRL over f-IRL is more pronounced even on some of the low-dimensional tasks, because RHIRL accounts for the control noise explicitly in the cost function.



Drive in the center of the lane. Adjust the direction at a sharp turn. Choose the shortest path to make the turn. Align to the lane center after the turn.

Figure 4: Driving behaviors learned by RHIRL for CarRacing-v0, with pixel-level raw image input.

Next we evaluate RHIRL and other methods for robustness under noise. A robust cost function encodes the expert's true intent. It is expected to yield consistent performance over different noise levels, regardless of noise in expert demonstrations.

For each task, a cost function is learned in a noise-free environment and is then used to re-optimize a policy in the corresponding noisy environments. Specifically for GAIL, since it learns a policy and does not recover the associated cost function, we directly apply the learned policy in noisy environments.

4.3 Robustness

Table 2 shows that noise causes performance degradation in all methods. However, RHIRL is more robust in comparison. For simple tasks, Pendulum and Lunarlander, RHIRL and f-IRL perform consistently well across different noise levels, while GAIL and AIRL fail to maintain their good performance, when the noise level increases. For the more challenging tasks, Hopper and Walker, RHIRL's performance degrades mildly, and f-IRL suffers more significant performance degradation. It is worth noting that the expert demonstrations used to train the transferred cost function are from the perfect system. Therefore, some expert actions and states may no longer be optimal or feasible in a highly noisy environment. Moreover, the cost function trained in the perfect system cannot reason about the long-term consequences of an action in a high noise environment. Therefore, it is challenging for the learned cost function to be robust to a highly noisy environment, as capturing the true intention of the expert is difficult.

Table 2: Robustness of RHIRL and other methods under noise. The cost functions are learned in noise-free environments and evaluated in noisy environments. The performance is measured as the ratio between the average return of an re-optimized policy in a noisy environment and the expert's average return in the corresponding noise-free environment. The absolute average returns and the standard deviations are reported in Appendix D.4. The negative ratios are clipped to 0.

		Noise Level Σ		
		0.0	0.2	0.5
Pendulum	Expert	-154.69 $\pm$ 67.61	–	–
3, 1	RHIRL	1.06	**1.07**	**1.06**
	f-IRL	**1.08**	0.90	0.85
	AIRL	1.05	0.79	0.67
	GAIL	0.88	0.71	0.62
LunarLander	Expert	232.00 $\pm$ 86.12	–	–
8, 2	RHIRL	**1.05**	**0.89**	**0.76**
	f-IRL	0.76	0.53	0.44
	AIRL	0.74	0.14	0.10
	GAIL	0.72	0.44	0.34
Hopper	Expert	3222.48 $\pm$ 390.65	–	–
11, 3	RHIRL	0.95	**0.80**	**0.67**
	f-IRL	**0.96**	0.65	0.62
	AIRL	0.01	0.01	0.00
	GAIL	0.82	0.07	0.06
Walker	Expert	4999.47 $\pm$ 55.99	–	–
17, 6	RHIRL	**0.99**	**0.80**	**0.69**
	f-IRL	**0.99**	0.60	0.22
	AIRL	0.00	0.28	0.36
	GAIL	0.50	0.02	0.02
Ant	Expert	5759.22 $\pm$ 173.57	–	–
111, 8	RHIRL	**0.86**	**0.55**	**0.15**
	f-IRL	0.87	0.35	0.08
	AIRL	0.17	0.15	0.00
	GAIL	0.48	0.00	0.00
CarRacing	Expert	903.25 $\pm$ 0.23	–	–
96×96, 3	RHIRL	**0.40**	**0.29**	**0.12**
	f-IRL	0.09	0.02	0.00
	AIRL	0.00	–	–
	GAIL	0.00	–	–

4.4 Effect of Receding Horizon K

The receding horizon K allows RHIRL to trade off optimality and efficiency. To study its effect, we use Pendulum with $K \in \{5, 20, 30, 75\}$. When K is small, RHIRL learns fast with fewer environment steps, but may be unstable. For $K = 5$, the performance jumps after the first few steps, but fluctuates drastically, as K is much smaller than the number of look-ahead steps for the pendulum to swing up. The learned cost function tends to overfit to local behaviors and forget the useful reward signals learned previously. With larger K values (20 or 30), the performance of the cost function learned is more stable, without much loss in efficiency. However, when K is too large, e.g., $K = 75$, RHIRL becomes less efficient and

takes longer to converge, because that excessive K values enlarge the variance of sampled control sequences.

5 Conclusion

RHIRL is a scalable and robust IRL algorithm for high-dimensional, noisy, continuous systems. Our experiments show that RHIRL outperforms several leading IRL algorithms on multiple benchmark tasks, especially when expert demonstrations are noisy.

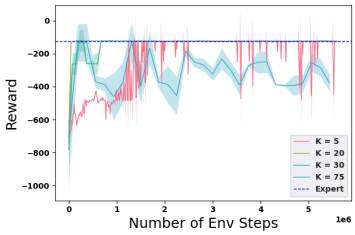


Figure 5: The effect of receding horizon K on the performance of RHIRL on the Pendulum task.

RHIRL's choice of local rather than global optimization is an important issue that deserves further investigation. Overall, we view this as an interesting trade-off between scalability and optimality. While this trade-off is well known in reinforcement learning, optimal control, and general optimization problems, it is mostly unexplored in IRL. Further, local optimization may tie the learned cost with the optimizer. It would be interesting to examine whether the learned cost transfers easily to other domains with different optimizers. We are keen to investigate these important issues and their implications to IRL as our next step.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

[2] Ulrich Viereck, Andreas Pas, Kate Saenko, and Robert Platt. Learning a visuomotor controller for real world robotic grasping using simulated depth images. In *Proc. Conference on Robot Learning*, 2017.

[3] A.Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. Int. Conf. on Machine Learning*, 1999.

[4] W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis) design for autonomous driving. arXiv:2104.13906, 2021.

[5] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proc. Int. Conf. on Machine Learning*, 2000.

[6] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2008.

[7] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning, 2016.

[8] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proc. Int. Conf. on Machine Learning*, 2016.

[9] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *Proc. Int. Conf. on Learning Representations*, 2018.

[10] Kyriacos Shiarlis, João V. Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *AAMAS*, 2016.

[11] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6818–6827. PMLR, 09–15 Jun 2019.

[12] Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations, 2019.

[13] W.B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience, 2007.

[14] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proc. Int. Conf. on Artificial Intelligence & Statistics*, 2011.

[15] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proc. Int. Conf. on Artificial Intelligence & Statistics*, 2011.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2014.

[17] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016.

[18] Ahmed H. Qureshi, Byron Boots, and Michael C. Yip. Adversarial imitation via variational inverse reinforcement learning. In *Proc. Int. Conf. on Learning Representations*, 2019.

[19] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In *Proc. Conference on Robot Learning*, 2019.

[20] Tianwei Ni, Harshit Sikchi, Yufei Wang, Tejus Gupta, Lisa Lee, and Benjamin Eysenbach. f-irl: Inverse reinforcement learning via state marginal matching. In *Proc. Conference on Robot Learning*, 11 2020.

[21] Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric P. Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. 2019.

[22] Letian Chen, Rohan R. Paleja, and Matthew C. Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. *Proc. Conference on Robot Learning*, 2020.

[23] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning, 2018.

[24] Hana Hoshino, Kei Ota, Asako Kanezaki, and Rio Yokota. Opirl: Sample efficient off-policy inverse reinforcement learning via distribution matching, 2021.

[25] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336, 2013.

[26] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 475–482, Madison, WI, USA, 2012. Omnipress.

[27] Wook-Hyun Kwon and S Han. *Receding Horizon Control: Model Predictive Control for State Models*. Springer-Verlag London, 01 2005.

[28] James MacGlashan and Michael L. Littman. Between imitation and intention learning. In *IJCAI*, pages 3692–3698, 2015.

[29] Keuntaek Lee, David Isele, Evangelos A. Theodorou, and Sangjae Bae. Spatiotemporal costmap inference for mpc via deep inverse reinforcement learning, 2022.

[30] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *Proc. Int. Conf. on Machine Learning*, 2019.

[31] Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Proc. Conference on Robot Learning*, 2019.

[32] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.

[33] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models, 2016.

[34] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2017.

[35] Grady Williams, Brian Goldfain, Paul Drews, Kamil Saigol, James M. Rehg, and Evangelos A. Theodorou. Robust sampling based model predictive control with sparse objective information. *Robotics: Science and Systems XIV*, 2018.

[36] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[37] David Ha and Jürgen Schmidhuber. World models. *CoRR*, 2018.

[38] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[39] Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization, 2021.

[40] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. Feedback in imitation learning: The three regimes of covariate shift, 2021.

[41] Evangelos A. Theodorou and Emanuel Todorov. Relative entropy and free energy dualities: Connections to path integral and kl control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 1466–1473, 2012.

[42] Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, USA, 2010. AAI3438449.

[43] Nan Jiang, A. Kulesza, S. Singh, and R. Lewis. The dependence of effective planning horizon on model accuracy. volume 2, pages 1181–1189, 05 2015.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes] See Section 5

(c) Did you discuss any potential negative societal impacts of your work? [N/A]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes]

(b) Did you include complete proofs of all theoretical results? [Yes] See Appendix

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [N/A]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]