Cost Function Estimation Using Inverse Reinforcement Learning with Minimal Observations

Sarmad Mehrdad[1], Avadesh Meduri[1] and Ludovic Righetti[1,2]

***Abstract*— We present an iterative inverse reinforcement learning algorithm to infer optimal cost functions in continuous spaces. Based on a popular maximum entropy criteria, our approach iteratively finds a weight improvement step and proposes a method to find an appropriate step size that ensures learned cost function features remain similar to the demonstrated trajectory features. In contrast to similar approaches, our algorithm can individually tune the effectiveness of each observation for the partition function based on the current estimate of the cost function parameters, guiding the algorithm towards better estimates in the following iterations. In addition, it does not need a large sample set, enabling faster learning. We generate sample trajectories by solving an optimal control problem instead of random sampling, leading to more informative trajectories. The performance of our method is compared to two state of the art algorithms to demonstrate its benefits in several simulated environments.**

I. Introduction

Inverse Reinforcement Learning (IRL) and Inverse Optimal Control (IOC) both aim to extract the cost function of a task given the observation(s) of an "expert" or "optimal" task execution [1]. The problem of extracting a cost function from expert observations has several benefits. Primarily, it goes beyond explaining the expert behavior to one environment in a particular task, and tries to explains *why* the task was executed in the observed manner. Moreover, this allows for a generalized behavior of the system facing various environments and inevitable external inputs where simply *mimicking* the expert cannot be fruitful. Because of this, IRL has become attractive for researchers in various fields such as optimal control [2], human intent inference [3], human movement analysis [4] and humanoids [5] (cf. [6], [7] for extended reviews of the topic).

However, extracting an optimized cost function given an expert demonstration is challenging at best. The most important challenge is that there exists multiple cost functions that can explain the observed expert movement, making IRL an ill-posed problem full of local optima. This is in contrast to reinforcement learning (RL) or optimal control (OC), where the cost and task space information is readily available and therefore understood easily. It is also fairly complicated for IRL to sample non-optimal demonstrations for training, and providing expert examples for IRL is not always feasible.

Prior work have tried to solve the IRL problem by either using a maximum margin formulation [8], or considering a task related probability distribution to quantify the occurrence of the expert demonstration with highest probability given the cost function [9] using maximum entropy principle [10]. Initially, in [9], for small discrete domains dynamic programming was used to estimate the distribution accurately. Further in [11], this approach was modified to optimize the embedded maximum likelihood estimation given the relative entropy of the samples. Since then, efforts have been made to generalize these method to higher dimensions and continuous domains using various techniques for estimating the distribution such as approximation [12] and various sampling methods [13]–[15].

Although these approaches have shown great potential for learning the cost function, they still suffer from various shortcomings. For example, local sampling around expert demonstration, as used in [13], [14], [16], might fail to contain informative and diverse non-optimal demonstrations to represent task space as a whole. Furthermore, requiring initial sample set can be challenging as well. It either necessitates to use specialized sampling techniques, or to use simpler uniform sampling while risking inaccuracy. Aside from the need for initial samples, most approaches require to use all the samples iteratively for accurate estimation of the trajectory space, which can be computationally demanding. In addition, a majority of IRL methods tend to control convergence based on probabilistic features such as maximum or relative entropy. Hence, the cost function will not be estimated by how optimal it can generate a new trajectory close to the expert demonstration. Some of these issues can be mitigated with IOC approaches, where gradient-based methods are used to solve the problem, hence foregoing the need of trajectory sampling. However, IOC approaches tend to be computationally expensive as they require to solve a bi-level optimization problem which includes an OC problem in its inner loop [5].

Another challenge in regards to approximating the trajectory space in the sampling-based approaches, is to incorporate *how* non-optimal is a sampled trajectory within the trajectory set. To capture this measure of non-optimality, in [13], the trajectories are sampled in the local vicinity of the optimal trajectory, considering their non-optimality as uniform within that vicinity. Closer to our approach, in [11] and [15], non-optimal samples are individually rescaled. While these efforts improve the trajectory space approximations, they do not rely on the policy parameters the IRL is optimizing, but rather on the system dynamics uncertainties and state-action transition probability distributions.

To address these challenges, this paper proposes an it-

This work was in part supported by the National Science Foundation grants 2026479, 2222815 and 2315396 and Wandercraft.

[1]Tandon School of Engineering, New York University, USA

[2]Artificial and Natural Intelligence Toulouse Institute (ANITI), France

erative sampling-based IRL algorithm based on maximum entropy principles for continuous task spaces. The approach seeks to iteratively find improvements in the cost estimate while ensuring it would generate trajectories close to the demonstration. We reformulate the probability distribution of the demonstrations by individually tuning the effectiveness of each sample based on the the previous iteration's estimate of the cost function parameters.

Moreover, sampling is conducted using an OC solver, where at each iteration we generate new trajectories based on the current cost function estimation. This generates meaningful samples consistent with the the intended use of the cost function and relaxes the need for an ad-hoc sampling strategy. We demonstrate empirically that our approach leads to significantly faster convergence and higher quality estimates when compared to state of the art methods.

The remainder of this paper is structured as follows. In Section II we present a brief preliminary to our work, followed by full description of our method in Section III. We present the results for our method in Section IV, and discussions in Section V.

II. Preliminaries

Our method relies on the probabilistic approach of [9], which writes the probability of the expert's demonstration (referred to from now on as the optimal trajectory)

$$P(\tau^\star|\bar{\boldsymbol{\tau}}) = \frac{1}{Z}\exp(-C(\tau^\star)) \tag{1}$$

with *partition function*

$$Z = \exp(-C(\tau^\star)) + \sum_{\tau_i \in \bar{\boldsymbol{\tau}}} \exp\left(-C(\tau_i)\right) \tag{2}$$

where $\tau^\star$ is the optimal trajectory. Observed trajectories are denoted as $\tau = \{\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots, \boldsymbol{x}_T\}$ where $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ are state and control inputs at time t. T is the length of the trajectory. $C(\tau)$ is the cost of trajectory τ, and $\bar{\boldsymbol{\tau}} = \{\tau_1, \tau_2, \ldots, \tau_K\}$ is the set of all observed trajectories, excluding $\tau^\star$. With this formulation, trajectories with higher cost are exponentially less likely to occur and vice versa.

The optimal expert demonstration $\tau^\star$ should then have the highest probability compared to all other possible trajectories. Hence, IRL approaches based on (1) aim to maximize the probability of the optimal trajectory, under any set of trajectory observations $\bar{\boldsymbol{\tau}}$

$$C^\star = \arg\min_C -\log(P(\tau^\star|\bar{\boldsymbol{\tau}})) \tag{3}$$

A key challenge in such methods is estimating the partition function, i.e. generating a representative set of non-optimal trajectories that will help discriminate cost functions when solving (3). In the following we propose a method for generating trajectories used in the partition function Z that allows the IRL algorithm to have a reasonable understanding of how much each observed trajectory should be incorporated into the partition function. This approach allows for a better representation of the partition function, and even relaxes the need to have all observed trajectories used for the trajectory space representation, hence reducing computational cost.

III. Methods

As mentioned above, we are interested in finding cost functions represented by a linear combination of explicit features. This representation is more constrained than IRL approaches based on neural networks but has two benefits: it allows a seamless integration of the learned cost with optimal control solvers for model-predictive control (MPC) and provides interpretable cost functions, which is important for human movement understanding or human-robot interaction tasks. Further, such cost functions are capable of generating diverse movements and achieving many tasks [17]. We hence search for costs represented as a weighted sum of features

$$C(\tau, \boldsymbol{w}) = \boldsymbol{w}^T \Phi(\tau) \tag{4}$$

$$\Phi(\tau) = \sum_{t=0}^{T-1} \phi_s(\boldsymbol{x}_t, \boldsymbol{u}_t)\Delta t + \phi_T(\boldsymbol{x}_T) \tag{5}$$

where $\Phi(\tau)$ is a vector of time integrated features of τ. ϕ_s and ϕ_T are vectors of stage and terminal cost features, respectively. For brevity, the features of the i^{th} trajectory is denoted Φ_i, and $\Phi^\star$ is the optimal trajectory's feature vector.

A. Problem Statement

Given the linearity assumption of the cost function, the IRL problem becomes

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} -\log(P(\tau^\star|\boldsymbol{w}, \bar{\boldsymbol{\tau}})) \tag{6}$$

$$\text{s.t.} \quad \boldsymbol{w} \geq \boldsymbol{0} \tag{7}$$

This problem aims to find weights that maximize the probability of the optimal trajectory for a set of observed trajectories. As a consequence, the recovered weights should then produce the same optimal trajectory when used in an OC solver. However, maximizing the probability of the optimal trajectory does not guarantee this. Indeed, the partition function cannot be computed for all possible trajectories and will be computed from a finite, often small, set of non optimal trajectories. Therefore, there exist trajectories not included in the set that can have higher probability than the demonstration. This issue is typical of all IRL approaches based on (1). In this paper, we mitigate this issue by proposing a way to minimize the difference between the estimated costs and the features of the optimal ($\Phi^\star$) and OC trajectory (denoted henceforth as $\tilde{\Phi}$).

We propose to solve the problem iteratively by taking small steps Δw to improve the cost candidate and propose a procedure to accept such a step at each iteration to ensure it also improves the trajectory that will be generated by the OC solver.

B. Computing a step direction

We can explicit the minimizing of (6) as

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} -\log \frac{1}{1 + \sum_{\tau \in \bar{\boldsymbol{\tau}}} e^{-\boldsymbol{w}^T(\Phi_i - \Phi^\star)}} \tag{8}$$

Since we seek to find an improvement step, we write the weights at iteration $n+1$ as $\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \Delta \boldsymbol{w}_n$ which leads to the following problem

$$\Delta \boldsymbol{w} = \arg\min_{\Delta \boldsymbol{w}} -\log \frac{1}{1 + \sum_{\tau \in \bar{\boldsymbol{\tau}}} \gamma_i e^{-\Delta \boldsymbol{w}_t^T (\Phi_i - \Phi^\star)}} \tag{9}$$

$$\text{s.t.} \quad \Delta \boldsymbol{w}_t > -\boldsymbol{w}_t \tag{10}$$

where

$$\gamma_i = e^{-\boldsymbol{w}_t^T (\Phi_i - \Phi^\star)} \tag{11}$$

can be understood as a weight on each trajectory to give more importance to non-optimal trajectories that are more probable. This means that trajectories that are farther from optimality will weigh less on the 'changes' in weights and vice versa. The benefits of assigning a weight to each trajectory when computing the partition function has already been discussed in [11], [15], where an importance sampling argument was used to specifically compute these weights. In our approach, these weights, which are dependent on both trajectory features and cost weights, appear naturally as we search for an improvement direction Δw.

C. Step Acceptance Method

Once a step candidate $\Delta \boldsymbol{w}$ has been computed by solving problem (9), we need to determine the actual step size to ensure that the change of weight will help find a cost function that will lead to trajectories closer to the optimal one. Reminiscent to line search methods in optimization, we generate a trajectory with an OC solver using $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \Delta \boldsymbol{w}_t$ where $\alpha = 1$, and check if this trajectory is closer to the optimal trajectory in either feature space or cost using the following two merit functions

$$m_1(\boldsymbol{w}_{t+1}) = m_1(\boldsymbol{w}_t + \alpha \Delta \boldsymbol{w}) = \frac{1}{2}(\boldsymbol{w}^T \Phi^\star - \boldsymbol{w}^T \tilde{\Phi})^2 \tag{12}$$

$$m_2(\boldsymbol{w}_{t+1}) = m_2(\boldsymbol{w}_t + \alpha \Delta \boldsymbol{w}) = ||\Phi^\star - \tilde{\Phi}||_2 \tag{13}$$

We will accept a step if it leads to an improvement on either of these merit functions. For m_1, as we can compute its derivative with respect to Δw, we use Wolfe conditions to measure acceptable improvement. For m_2 we simply require that it is lower than its value at the previous iteration. We then add the OC solver's resulting trajectory with the new weight estimate to $\bar{\boldsymbol{\tau}}$ and iterate again. Otherwise, we decrease α and try again. In our algorithm, if we did not find an acceptable step-size after 10 iterations, the algorithm terminates.

The procedure above also explains our approach to sampling, as we add trajectories computed with an OC solver in our dataset. Initially, the algorithm has bad weight estimation and it results in trajectory samplings that are far from optimal. As iterations proceed, generated trajectories become closer to the optimal one. This way, our trajectory sample set will contain both "good" and "bad" non-optimal trajectories.

D. Sub-sampling Trajectories

We propose to augment our dataset by adding trajectories of reduced length from the current trajectories. For each full length trajectory, with T timestamps per trajectory, in the partition function, we create a sub trajectory starting from $t = d$ and ending at $t = T$. We generate several subset of trajectories by using different values of $d \in D$. Our minimization is then adapted to

$$\Delta \boldsymbol{w} = \arg\min_{\Delta \boldsymbol{w}} \sum_{d \in D} -\theta_d \log \frac{1}{1 + \sum_{\tau \in \bar{\boldsymbol{\tau}}} \gamma_i e^{-\Delta \boldsymbol{w}^T (\Phi_{i,d} - \Phi_d^\star)}} \tag{14}$$

where $\Phi_{i,d}$ and $\Phi_d^\star$ denote features truncated from $t = d$ and D is the set of timestamps equidistantly chosen between 0 and T. For example, for $N = 3$, in addition to the original set we will have two more subsets, containing trajectories with a one-third and two-thirds of their original lengths, i.e., if $T = 100$, then $D = \{0, 33, 66\}$. To give more importance to longer trajectories, we scale each subset's likelihood by a coefficient $\theta_d = (T - d + 1)/(T + 1)$. This can be useful for lower dimensional feature spaces, and lower length trajectories where the trajectory cost can be less sensitive to weight changes.

E. Regularization

We also add L1 and L2 regularization to form an *elastic net* regularization, to lower weights magnitude and favor sparsity. The induced sparsity in $\Delta \boldsymbol{w}$ causes the most important features to change weight instead of all at the same time. We thus minimize

$$\Delta \boldsymbol{w} = \arg\min_{\Delta \boldsymbol{w}} \sum_{d \in D} -\theta_d \log \frac{1}{1 + \sum_{\tau \in \bar{\boldsymbol{\tau}}} \gamma_i e^{-\Delta \boldsymbol{w}^T \bar{\Phi}_i d}} + \lambda |\Delta \boldsymbol{w}| + \frac{\beta}{2} ||\Delta \boldsymbol{w}||_2^2 \tag{15}$$

Regularization is useful when feature space is small, and learning process can be challenging especially when opposing features are present as seen in the experiments below.

F. Moving Window

Since our approach restricts the steps taken to improve the weights and further scales the importance of each trajectory in the partition function individually, it is not necessary to keep a large history of trajectories to compute the partition function. Indeed, trajectories far from the optimal one will have a negligible weight. Therefore we will only use the last L trajectories at each iteration, discarding older trajectories.

This approach is in contrast to several maximum entropy based IRL algorithms which typically require many trajectories to work well. This further helps reduce computational cost and empirically we notice that it leads to faster convergence as can be seen in Sections IV-A.2 and IV-B.2.

The method is summarized in Algorithm 1. To initialize the trajectory set $\bar{\tau}$, we solve the OC problem given an initial guess of cost weights which we set to small non-zero values. The resulting trajectory τ_0 is the initial set of the non-optimal trajectories. $\bar{\boldsymbol{\tau}}_{all}$ represents the set of all sampled trajectories throughout the iterations.

IV. Results

In this section, we evaluate our approach (MO-IRL for Minimum Observation IRL) on two different environments and compare it with state of the art approaches. The first

Algorithm 1 MO-IRL

Input: $\tau^\star, L, N$
Output: $\boldsymbol{w}_{irl}$
$t \leftarrow 0$, $\boldsymbol{w}_0 \leftarrow [\mathbf{0.01}]$ # Non-zero small values
$D = \{0, \frac{1}{N}T, \frac{2}{N}T, \dots \frac{N-1}{N}T\}$ # for N sub-samples
$\tau_0 = OC(\boldsymbol{w}_0)$, $\bar{\boldsymbol{\tau}} = \{\tau_0\}$, $\bar{\boldsymbol{\tau}}_{all} = \bar{\boldsymbol{\tau}}$
$M_1 \leftarrow \infty$, $M_2 \leftarrow \infty$
while not converged **do**
  Find $\Delta \boldsymbol{w}$ using (15)
  $\alpha \leftarrow 1$
  while Step not found for 10 trials **do**
    if $m_1(\boldsymbol{w}_t + \alpha\Delta\boldsymbol{w}) < M_1$ or $m_2(\boldsymbol{w}_t + \alpha\Delta\boldsymbol{w}) < M2$ **then**
      $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha\Delta\boldsymbol{w}$ # Step Found
      $M_1 \leftarrow m_1(\boldsymbol{w}_t + \alpha\Delta\boldsymbol{w})$
      $M_2 \leftarrow m_2(\boldsymbol{w}_t + \alpha\Delta\boldsymbol{w})$
    else
      $\alpha \leftarrow \alpha/4$ # Step Not Found
    end if
  end while
  $\tau_{t+1} = OC(\boldsymbol{w}_{t+1})$
  $\bar{\boldsymbol{\tau}}_{all} = \{\tau_0, \tau_1, \dots, \tau_{t+1}\}$
  if $t + 1 > L$ **then**
    $\bar{\boldsymbol{\tau}} = \bar{\boldsymbol{\tau}}_{all}$
  else
    $\bar{\boldsymbol{\tau}} = \{\tau_{t-L+1}, \tau_{t-L+2}, \dots, \tau_{t+1}\}$
  end if
  $t \leftarrow t + 1$
end while

environment is a simple goal-reaching and obstacle avoidance task with a point mass model. The second environment is a reaching task with obstacle avoidance with a simulated Kuka IIWA 14 robot. We compare our results against the approaches proposed in [16] and [14], which will henceforth be referred to as PI^2-IRL and IS-IRL (for Iterative Scaling), respectively. These methods sample local trajectories based on the method provided in [18]. We used Pinocchio [19] together with the Croccoddyl framework [17] and the MiM_Solver nonlinear SQP solver [20] to solve OC problems. We used MuJoCo [21] for the robot simulation. The demonstrated trajectory in all subsequent examples is generated by setting cost weights and using the SQP solver. We do not use the optimal weights anywhere in the tests.

A. Point Mass Environment

1) Problem setup: Our first environment is a 2D point mass that reaches a target reaching and obstacle avoidance. The task is to reach the green square, while avoiding the gray circles. The full trajectories are $1.5s$ long for in the one obstacle case, and $2.5s$ seconds long for the other cases with $\Delta t = 0.05$. We use the following cost features: Goal tracking ('G') is a quadratic cost on both state distance to the goal and non-zero velocity, state and control regularization (Namely 'XReg' and 'UReg') is the squared summation of the corresponding vectors, obstacle avoidance cost for N_{obs} obstacles ('Obs') is zero if the point mass is farther from an obstacle by an activation margin l, and otherwise increases quadratically with the distance to the obstacle's center O_i. We use the same set of features (excluding UReg) for the terminal costs.

The initial trajectory set for PI^2-IRL and IS-IRL were generated by rolling out 20 noisy control inputs in the parametrization of the optimal trajectory. For MO-IRL, however, there is no need for an initial set, and only an OC solution with small non-zero weights will suffice as the 'set' of non-optimal trajectories. For the moving window of the MO-IRL we used $L = 1$, meaning that at each iteration only the previously generated trajectory is used for updating the weights. Lastly, we used $N = 20$ sub-samples for the trajectories, and regularized the maximum likelihood estimation by choosing $\lambda = 10^{-6}, \beta = 10^{-2}$.

2) Results: Fig.1 shows the performance of MO-IRL in comparison with PI^2-IRL and IS-IRL. We test the algorithms in 3 different point mass environments with 1, 4, and 5 obstacles (referred to as 'PM1', 'PM2', and 'PM3', respectively), and show the OC solution for the resulting cost function for the point mass starting from different locations on the map. For MO-IRL, convergence happens after 6, 2, and 2 iterations for PM1, PM2, and PM3 respectively, with samplings getting closer to the optimal trajectory faster than the other methods. We can also see, that with changes in the task (starting from other points) the point mass still avoids obstacles and reaches the goal.

PI^2-IRL however, fails to learn a cost that avoids obstacles. This mismatch of goal association comes from the fact that 'XReg' and 'G' features are inherently opposing each other, making it a challenge to tune them well. During optimization, we observed that PI^2-IRL fails to lower the 'XReg' weights in the terminal features, which results in the attraction of the point mass towards the starting point when approaching the terminal state. IS-IRL samples trajectories at each iteration by analyzing the entire sampled set, and based on the minimum cost difference of the trajectories, increases the *temperature* of the exponentiated cost in the maximum likelihood estimation process. This tends to render learning more sensitive to smaller changes in the feature space as samples get closer to optimality. Although this approach has the advantage of monitoring the IRL improvement, it has some potential issues. First, the stopping criteria only rests on temperature passing a threshold, and not how the weights converge. As a result, IS-IRL will increase the weights to much higher values than needed by the last iterations, losing sensitivity to some features, which can lead to some issues. For example, we see in Fig.1 that in PM2, all trajectories starting behind the obstacles converge to a local minimum and never reach the goal. Indeed, the loss of sensitivity to certain features led the algorithm to give too large weight on the obstacle avoidance costs to the detriment of target reaching. Secondly, IS-IRL intends to change the effectiveness of the sampled trajectories, but it does so by changing the temperature for the whole set, meaning that the entire set is needed to form the partition function. This can be computationally heavy as iterations proceed. Conversely, in our method the influence of the trajectories is individually tuned given the previous weights, which relaxes the need for having the full set at all times.

Fig.2 shows the performance of the algorithms on the point mass tests. MO-IRL concludes the training much earlier than IS-IRL, even though it terminates on a higher trajectory costs. IS-IRL usually starts the iterations where PI^2-IRL finishes the learning, which is expected since without iterative scaling of the partition function, IS-IRL and PI^2-IRL are practically very close in nature. Table I, shows the costs of the trajectories generated from the learned weights from each algorithm using the original optimal cost function. Our approach produces a trajectory with a cost closer to the optimal cost. In addition, computation time is provided in Table II and stresses the performance of the method to find a suitable cost function. While PI2-IRL is faster, it does not converge to good cost functions. We implemented each algorithm in a similar way but our results need to be taken with care as it is always possible to improve an implementation.

To better understand the importance of each part of the algorithm, we conducted experiments with PM2 with versions of the algorithm that did not include the step acceptance strategy, the regularization or sub-sampling. The results are summarized in Fig. 3. We notice that without either of these approaches, the algorithm fails completely at recovering a good cost. Adding sub-sampling leads to some trajectories reaching the target. Adding the step acceptance strategy further leads to trajectories that always reach the target several are colliding with the obstacles. Regularization similarly improves the behavior of the algorithm, with similar obstacle collision issues. The best results are found when everything is used.

B. Robot Simulation

1) problem Setup and Initialization: In this example, we study a goal reaching task with obstacle avoidance for a simulated torque-controlled 7-DOFs Kuka iiwa robot. We use the same cost features (albeit in a higher dimensional space) and do the following modifications. The goal cost is computed in end-effector frame. We further create capsules around the last four links of the robot and use the distance between the capsules and the obstacles to compute obstacle feature costs (one feature per pair of capsule / obstacle).

TABLE I

POINT MASS COSTS GIVEN OPTIMAL COST FUNCTION

Cost $\times 10^4$	$\boldsymbol{w}^{\star T}\Phi_{MO}$	$\boldsymbol{w}^{\star T}\Phi_{IS}$	$\boldsymbol{w}^{\star T}\Phi_{PI^2}$	$\boldsymbol{w}^{\star T}\Phi^{\star}$
PM1	0.5085	0.5121	1.8025	0.5036
PM2	0.6061	6.0114^{+}	6.3888^{+}	0.5091
PM3	0.5613	0.5779	6.0938^{+}	0.4970

$^{+}$Did not converge to target.

TABLE II

POINT MASS IRL COMPUTATION TIME (SEC)

	PM1	PM2	PM3
MO-IRL	20.14 ± 3.60	24.69 ± 18.76	29.50 ± 9.51
IS-IRL	40.77 ± 3.54	93.07 ± 17.45	90.17 ± 8.79
PI^2-IRL	4.13 ± 3.07	4.72 ± 3.71	7.84 ± 2.91

*Results are achieved under the same setting on a single device, with an Intel Core i7-10750H CPU @ 2.60GHz

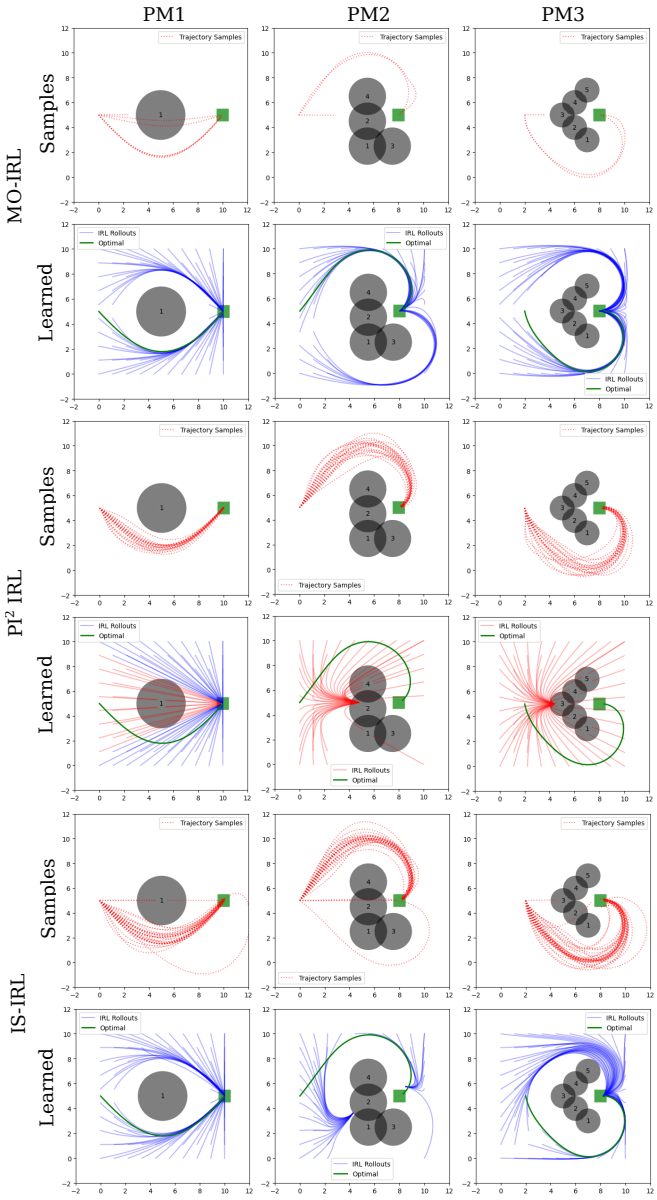

Fig. 1. The performance of the IRL algorithms on three point mass environments (one per column). Results are shown in pairs of rows for MO-IRL, PI^2-IRL, and IS-IRL. Top row of each pair shows the sampled trajectory set for each algorithm. Bottom rows of pairs show the set of OC rollouts based on the learned cost function. The initial optimal trajectory is shown in green in the bottom row. In the bottom rows, rollouts with obstacle collision are shown in red.

The full trajectories are one second long, with $\Delta t = 0.01$. The environment consists of $N_{obs} = 4$ box obstacles that the robot has to maneuver around in order to get to the green sphere target. For stage cost, there are 19 features (16 for obstacle avoidance, XReg, UReg, and G), and 18 for the terminal (All but UReg), making the total of 37 cost features. The initial trajectory set for PI^2-IRL and IS-IRL were generated by 20 noisy DMP rollouts. We noticed that when the feature numbers are relatively high, it is more beneficial to the learning if the weights are bounded as mentioned in III-D so that no *overweighting* would occur.

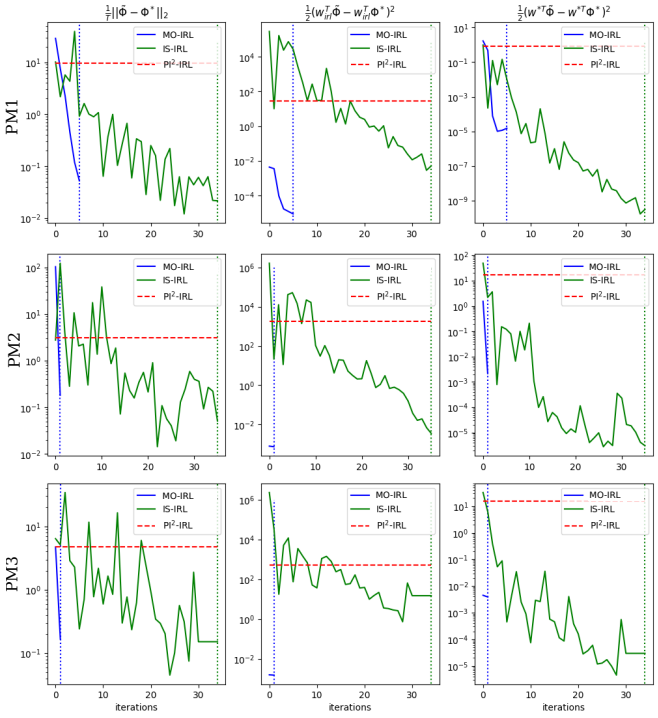

Fig. 2. Performance of MO-IRL and IS-IRL throughout their iterations. First column indicates the deviation of the resulting trajectory from optimality, second column shows the cost differences of the optimal and the solved trajectory given the estimated weight set, and the third column shows the cost difference of the optimal and solved trajectory given the optimal weights at each iteration. Vertical dashed lines show the termination of associated algorithms. PI^2-IRL's performance is shown by a horizontal red dashed line.

Although this weight binding is a helpful improvement for MO-IRL, it is necessary for IS-IRL since it weighs the obstacle avoidance features high and forgets other cost features, resulting in robot not moving at all. Hence, for the results presented in this section, we used weight bounds for all approaches ($\mathbf{1} \geq \boldsymbol{w}_t \geq \mathbf{0}$). In the robot simulation scenario, we choose parameters $L = 1$, $N = 20$, $\lambda = 10^{-6}$, and $\beta = 10^{-5}$ for MO-IRL.

2) Results: Fig.4 portrays the sampling set for IRL performances in the Kuka simulation. It can be seen that the trajectory sampling of MO-IRL is significantly different from that of the other approaches. The samples needed for MO-IRL are not local unlike the other samplings seen for PI^2-IRL and IS-IRL. In addition, due to this particular sampling, MO-IRL can converge with a lower number of trajectory samples. For PI^2-IRL the number of samples does not change (20 trajectories). For IS-IRL, this number rose to 55, and for MO-IRL, total sample count was only 14. Although the samples for MO-IRL are not as diverse as the ones for IS-IRL, there are trajectories that have gone through the obstacles that could not have been achieved by local rollouts. This ability of MO-IRL to generate informative bad demonstrations is extremely important for exposing the robot to states that it should learn not to go towards. For example, in the initial stage where there are only τ_0 and $\tau^\star$ sampled, MO-IRL optimize a cost that create trajectories going straight to the goal. This results in sampling a trajectory that hits the obstacles along its way, gaining more knowledge about the environment in subsequent iterations.

We compare the iterative performance of the algorithms for robot simulation in Fig.5. Similar to point mass tests, MO-IRL terminates after a lower number of iterations with higher optimal trajectory cost difference in comparison to IS-IRL. Unlike in the point mass tests however, MO-IRL produces the most optimal trajectory in terms of the features (left sub-figure). Moreover, since we used only the latest trajectory in MO-IRL, the computation for weight change is less demanding, and the convergence happened twice as fast as IS-IRL.

Fig. 6 shows how the robot conducted the task given the learned weights. PI^2-IRL in this test case performed poorly, as the same failure occurred for tuning the XReg against G, therefore robot stands mostly still. For that reason we cannot portray any movement resulting from PI^2-IRL in Fig. 6. MO-IRL and IS-IRL however showed good performance for the same task and when the goal position was changed. Apart from the environment with the same goal position (left sub-figure) where IS-IRL slightly grazes the lower obstacle, both algorithms showed nearly the same behavior.

To empirically investigate how the learned cost would fare in a different environment, we moved the obstacles and the goal in such a way that the robot needs to needle through the obstacles to get to the target. Fig. 7 shows the results for this experiment. Like in the previous environment, the robot shows performance with lower cost closer to the optimal one. However, we notice that the resulting behavior is different from the one generated with the optimal cost. The optimal cost (which we tuned for the first environment) generates a trajectory that hits the obstacles while the trajectory generated by our approach is different and does not hit the obstacles. We hypothesize that since our approach is designed to find small weight changes Δw, it tends to find smaller, well-balanced weights overall which might lead to better balanced feature selection. This will need, however, to be further investigated.

Table III shows the comparison between costs of the trajectory generated by the weights learned by MO-IRL, IS-IRL, and PI^2-IRL with respect to the desired cost function. We provide computational times to convergence as well as cost per iteration. MO-IRL is very efficient compared to IS-IRL (PI2-IRL did not find good costs).

TABLE III

ROBOT SIMULATION COSTS GIVEN OPTIMAL COST FUNCTION

Cost $\times 10^4$	$\boldsymbol{w}^{\star T}\Phi_{MO}$	$\boldsymbol{w}^{\star T}\Phi_{IS}$	$\boldsymbol{w}^{\star T}\Phi_{PI^2}$	$\boldsymbol{w}^{\star T}\Phi^\star$
Same Task	8.635	8.949	322.703^+	7.607
Changed Env.	365.221	394.249	380.1662^+	182.584
Duration (s)	20.8 ± 10.5	73.6 ± 28.5	45.9 ± 30.5	N/A
1 Iteration (s)	6.16 ± 5.11	2.07 ± 0.70	N/A	N/A

$^+$Did not converge to target.

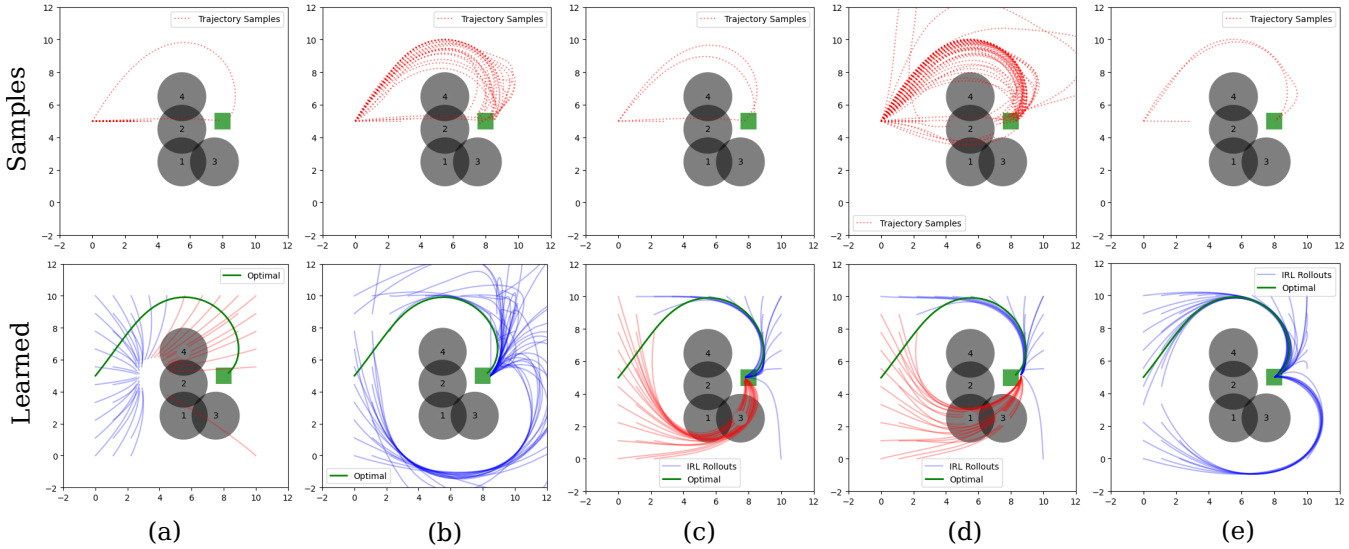

Fig. 3. Comparison of each improving step for modifying the IRL performance. **(a)** shows the algorithm without any step acceptance, regularization, and sub-sampling. **(b)** is the same as (a) with sub-sampling. **(c)** shows (b) with added step acceptance method. **(d)** is (b) but with regularization. **(e)** shows the full version with sub-sampling, regularization, and step acceptance method.

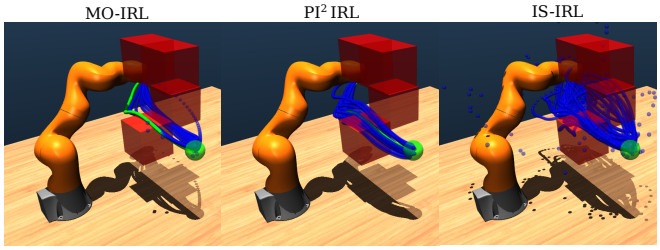

Fig. 4. Sampled trajectory sets for the IRL approaches. MO-IRL carries out a different way of sampling that is not necessarily local to the optimal trajectory. PI^2 IRL uses only the initial noisy local rollouts. IS-IRL concludes the iterations with a trajectory set containing both initial local rollouts, and OC's solutions. The green path seen on the left figure is the optimal trajectory.

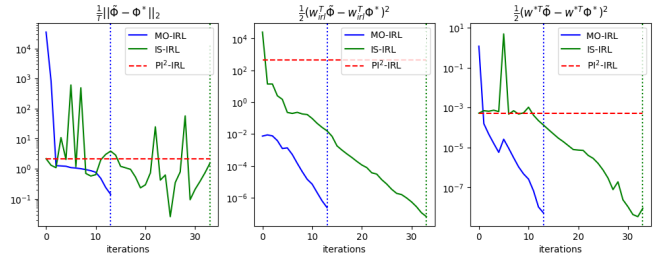

Fig. 5. Iterative performance comparison between MO-IRL and IS-IRL. PI^2-IRL's performance is indicated by the horizontal dashed red line.

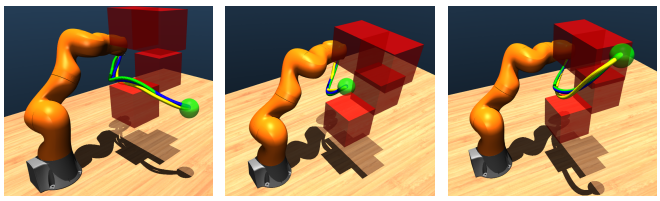

Fig. 6. Comparison of robot behavior in the same obstacle environment with different goal locations. MO-IRL, IS-IRL, and optimal solution given the desired weights, are shown in blue, yellow, and green respectively.

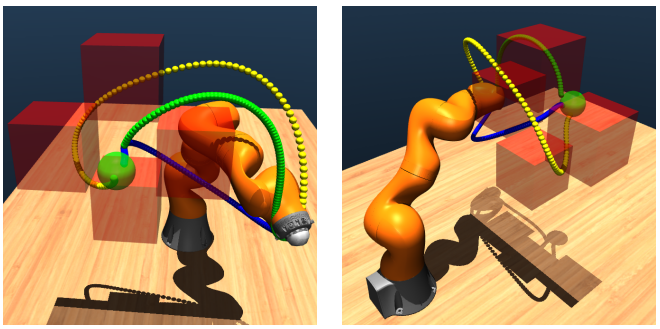

Fig. 7. Comparison of robot behavior in different challenging obstacle environment and goal location. The images are for the same test from different angles for better visibility. The color-coding is the same as in Fig. 6.

V. Discussions

We presented the MO-IRL algorithm, which iteratively improves the estimate of the optimal cost function by representing the trajectory space with minimal samples. We have shown that the approach can outperform state of the art IRL algorithms. Our method assigns individual tuning to trajectory samples for the partition function at each iteration depending on the most recent weight estimation with respect to the optimal trajectory. This means that any subset of samples in the set using this method can provide a reasonable space representation, relaxing the need for subjective tunings in the partition function, or necessity of including all samples. Due to the independence of MO-IRL to the whole sampled trajectory set, even one trajectory can be sufficient for learning the weights properly. This helps both convergence and computational complexity. Moreover, MO-IRL does not rely on local samplings and consequently does not require a policy parametrization. Instead, the samples are drawn from the OC solver at each iteration where a new cost

function is estimated.

In the future, we aim to exploit the algorithm and its advantageous computational complexity towards online cost learning and MPC implementations. This would be beneficial for human-in-the-loop scenarios where an expert is presently trying to interactively teach a task to a robot.

REFERENCES

[1] A. Y. Ng, S. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, no. 2, 2000, p. 2.

[2] A. Byravan, M. Monfort, B. D. Ziebart, B. Boots, and D. Fox, "Graph-based inverse optimal control for robot manipulation." in *Ijcai*, vol. 15, 2015, pp. 1874–1890.

[3] W. Liu, J. Zhong, R. Wu, B. L. Fylstra, J. Si, and H. H. Huang, "Inferring human-robot performance objectives during locomotion using inverse reinforcement learning and inverse optimal control," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2549–2556, 2022.

[4] A. M. Panchea, N. Ramdani, V. Bonnet, and P. Fraisse, "Human arm motion analysis based on the inverse optimization approach," in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. IEEE, 2018, pp. 1005–1010.

[5] K. Mombaur, A. Truong, and J.-P. Laumond, "From human to humanoid locomotion—an inverse optimal control approach," *Autonomous robots*, vol. 28, pp. 369–383, 2010.

[6] N. Ab Azar, A. Shahmansoorian, and M. Davoudi, "From inverse optimal control to inverse reinforcement learning: A historical review," *Annual Reviews in Control*, vol. 50, pp. 119–138, 2020.

[7] S. Adams, T. Cody, and P. A. Beling, "A survey of inverse reinforcement learning," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4307–4346, 2022.

[8] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.

[9] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[10] E. T. Jaynes, "Information theory and statistical mechanics," *Physical review*, vol. 106, no. 4, p. 620, 1957.

[11] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 182–189.

[12] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," *arXiv preprint arXiv:1206.4617*, 2012.

[13] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1331–1336.

[14] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 1561–1566.

[15] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International conference on machine learning*. PMLR, 2016, pp. 49–58.

[16] M. Kalakrishnan, E. Theodorou, and S. Schaal, "Inverse reinforcement learning with pi 2," in *The Snowbird Workshop, submitted to*. Citeseer, 2010.

[17] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2536–2542.

[18] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[19] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 614–619.

[20] A. Jordana, S. Kleff, A. Meduri, J. Carpentier, N. Mansard, and L. Righetti, "Stagewise implementations of sequential quadratic programming for model-predictive control," *Subm. IEEE TRO*, 2023.

[21] 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*. IEEE, 2012, pp. 5026–5033.