Training Robots to Evaluate Robots: Example-Based Interactive Reward Functions for Policy Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Physical interactions can often help reveal information that is not readily apparent. For example, we may tug at a table leg to evaluate whether it is built well, or turn a water bottle upside down to check that it is watertight. We propose to train robots to acquire such interactive behaviors automatically, for the purpose of evaluating the result of an attempted robotic skill execution. These evaluations in turn serve as "interactive reward functions" (IRFs) for training reinforcement learning policies to perform the target skill, such as screwing the table leg tightly. In addition, even after task policies are fully trained, IRFs can serve as verification mechanisms that improve online task execution. For any given task, our IRFs can be conveniently trained using only examples of successful outcomes, and no further specification is needed to train the task policy thereafter. In our evaluations on door locking and weighted block stacking in simulation, and screw tightening on a real robot, IRFs enable large performance improvements, even outperforming baselines with access to demonstrations or carefully engineered rewards. Project website: https://sites.google.com/view/lirf-corl-2022/

1 Introduction

Consider a kitchen robot that must perform a large number of tasks, such as opening a refrigerator, cutting vegetables, tightening a water bottle lid, or flipping a pancake. How might this robot acquire these skills? Common skill acquisition approaches involve heavy engineering and expertise for each skill, directed either towards developing model-based control policies, or towards specifying dense reward functions for reinforcement learning (RL). These approaches do not scale well to the goal of acquiring large numbers of skills. Consequently, there is growing interest in scalable approaches for RL-based skill acquisition that permit easy task specification by non-experts.

A particularly promising task specification framework [1–7] is exemplar rewards, in which the specification consists merely of showing the robot learner what the environment should look like after a well-executed skill. For example, to specify the task of opening a refrigerator door, it would suffice to show the learner some images of open refrigerators. Then, once the skill is specified, the robot performs RL with the aim of altering the environment to generate observations similar to those examples. When this framework is successful, a human user teaching the robot does not need technical expertise and does not need to even demonstrate full behaviors for task execution. Instead, they need only capture images of the outcome.

Although this framework is appealing, many tasks cannot be specified by image observations of task success.[1] For example, consider the task of closing a water bottle, as in Figure 1. What image might specify a successful execution of this skill? An image of a water bottle with a slightly loose cap would look identical to a tight cap, but a bottle closing policy that does not fully tighten the cap would be useless. Thus, in this setting, an observation of the task success state from a camera constitutes only a partial observation and does not suffice to specify the task. Specifically, when non-success

[1] or observations from any other fixed sensor setup

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

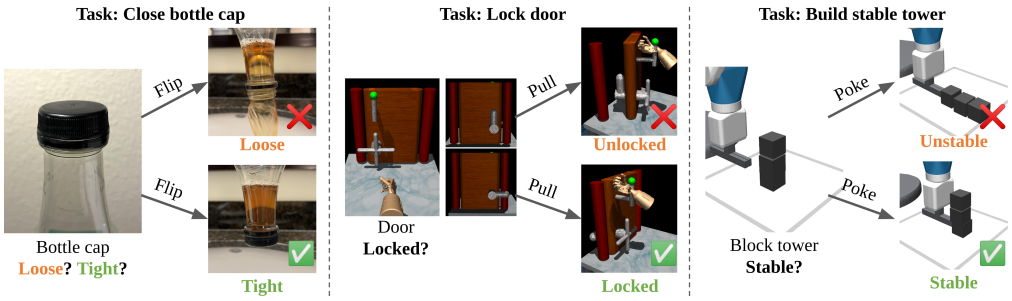

Figure 1: In each of these three settings, task success / failure cannot be determined from passive image observations, but is easily revealed through appropriate interactions.

states might look visually similar to success states, an image-based task specification fails. This state-aliasing problem hinders image-based task assessment in many other settings. Figure 1 shows some examples: Is the bottle cap tight? Is the door locked properly? Is the tower of objects with unknown masses stable? Other examples include: Is a screw tight? Is the leg of the table secured well? Does the smoothie have the right consistency? None of these questions can be answered correctly from passive image observations.

The failures of passive image-based assessment in these settings preclude robotic skill learning from image examples of task success. To fix this while retaining the versatility and user-friendliness of the exemplar rewards framework, we propose an *interactive* approach for evaluating task success. As motivation, consider how a human might turn a water bottle upside down to confirm that it does not leak, and is therefore closed well. Here, the interaction reveals the key determinant of task success, namely, whether or not the cap is watertight.

We design an approach to automatically learn such interaction behaviors to evaluate task success. In the water bottle setting, our examples are no longer mere image snapshots of closed water bottles, but instead *real actionable physical instances* of closed water bottles. The robot can interact with these objects, such as by turning them upside down, to discover what about them constitutes successful execution of the desired "bottle closing" skill (i.e., the bottle does not leak) and how they behave differently from the outcomes of unsuccessful execution (i.e., it leaks). Similarly, in the examples of Figure 1, a robot might try to pull the door open or nudge the tower to evaluate whether the task is complete.

We make two key contributions. First, we propose an approach to train *interactive reward functions*, which are policies that specify what it means to correctly perform a skill and thus drive the improvement of the task policy learners in a reinforcement learning framework. Next, we show how those same reward policies can be reused to enable *in-the-loop introspective verification* behaviors during online skill execution, making robots more scrupulous and reliable. We evaluate these contributions on three simulated and real robotic control settings, and show substantial improvements in task policy learning and execution.

2 Background and Problem Setup

2.1 Classifier-Based Exemplar Rewards in fully observed MDPs

Reinforcement learning (RL) methods offer the promise of scalable data-driven synthesis of robot controllers for arbitrary new tasks. Consider robotic task settings formalized as Markov decision processes (MDPs), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R)$. At each time t, an agent selects an action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$, transitions to the next state s_{t+1} with probability $\mathcal{T}(s_{t+1}|s_t, a_t)$, and receives reward $r_t = R(s_t, a_t, s_{t+1})$. Thus, the agent emits actions into the environment and receives two things in return: new state observations and rewards. Ignoring discounting, a good RL task policy $\pi_T(a_t|s_t)$ selects actions to maximize the sum of rewards over time $\sum_t r_t$.

Thus, it is the reward function R that specifies what task to perform in a given environment, and RL approaches in practice require expertise and effort for reward engineering [8] to specify each new task, avoiding misspecification and guiding efficient learning. This often takes the form of tuned

weighted combinations of many carefully constructed heuristic terms in the RL reward objective [9–12], or even privileged task-specific sensors added to the environment during training [13, 14]. To circumvent laborious manual reward design, many methods aim to learn rewards from data. Inverse reinforcement learning methods [15–18] learn task reward functions from optimal demonstrations, but such demonstrations are typically expensive and may even be impossible to obtain. Other methods train RL agents by learning rewards explicitly or implicitly from interactive human feedback [19–23], but these have the drawback of requiring in-the-loop queryable human teachers.

We build upon the popular "exemplar rewards" framework, which explores task specifications purely through examples [1–7]. In classifier-based exemplar rewards approaches [1, 3], the human teacher trains a classifier $\hat{R}(s)$ to label task outcomes as successes or failures, by gathering some success examples manually, and then generating failure examples from the task policy as it trains. Armed with this learned reward function $\hat{R}(s)$, the agent trains its task policy $\pi_T(a|s)$ to generate success outcomes and thereby maximize the estimated rewards. An appealing property of this framework is that training a policy for each new task only requires a specification of the final goal state that should be achieved, without any need to specify how to achieve it.

Our work extends this exemplar rewards framework: as motivated in Section 1, rather than relying on passive observations, we will train interactive evaluation policies to evaluate task success, and these evaluations will in turn serve as "interactive reward functions" for training task policies. This permits applying exemplar rewards to "partially observed" settings that do not permit accurate task success evaluation from images alone. We expand on the partial observability problem below.

2.2 Partial Observability, State Aliasing, and History-Based POMDP Policies

Why are classifier-based exemplar rewards approaches not suited to settings such as the door locking task in Figure 1 (middle)? The root cause is "partial observability." To solve this task, the robot must push the door into the closed position and then rotate the latch into the locking position. It only sees images from a fixed camera in front of the door. The latch, however, is occluded, and the robot can only observe and act upon the four handles on the front of the door, which rotate the latch through an axle mechanism. The handles are visually identical, so any visual configuration of the handles corresponds to any one of four configurations of the latch behind the door. While this setting is constructed as a pedagogical example to expose and study the state aliasing problem, it is representative of several real-world settings, as discussed in Section 1 and Figure 1.

Such settings are modeled as partially observable MDPs, or POMDPs [24]. At each time, the agent only partially observes the underlying latent task state s_t (i.e. the latch configuration), through an observation $o_t \sim O(o|s_t)$, where the observation function O generates samples from the observation space Ω (i.e, images of the handle configuration). The original task state s_t is not directly observable. Thus, the task POMDP is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \Omega, O)$.

Several prior works deal with the question of how the task policy in a POMDP might map to good actions despite only seeing partial observations at each time. The key to this is the assumption, very often reasonable, that while individual observations are indeed incomplete, observation *histories* contain much more information about the state. Then, a POMDP policy can perform well by relying on a history of observations and actions, either directly [25–28] as $\pi_T(a_t|o_{t-H:t}, a_{t-H-1:t-1})$, or by first estimating a belief distribution $b_t(s_t|o_{t-H:t}, a_{t-H-1:t-1})$ over the state and then mapping to actions [24, 29–32], as $\pi_T(a_t|b_t)$. In the door locking setting, the robot might observe the latch position before closing the door. Therefore, even if the latch is occluded after the door is closed, a history-based policy could still track its position. In other words, if trained well with the right reward function, such history-based POMDP policies can still learn good task behaviors.

3 Exemplar Interactive Reward Functions

This brings us to the key question: how to specify reward functions R for training POMDP policies? In particular, could we extend the appealing exemplar rewards framework to POMDPs?

First, note that partial observability has to do with what *policies* can see, not where *rewards* come from. Therefore, it is feasible to provide state-based rewards $R(s)$ that are not limited by partial

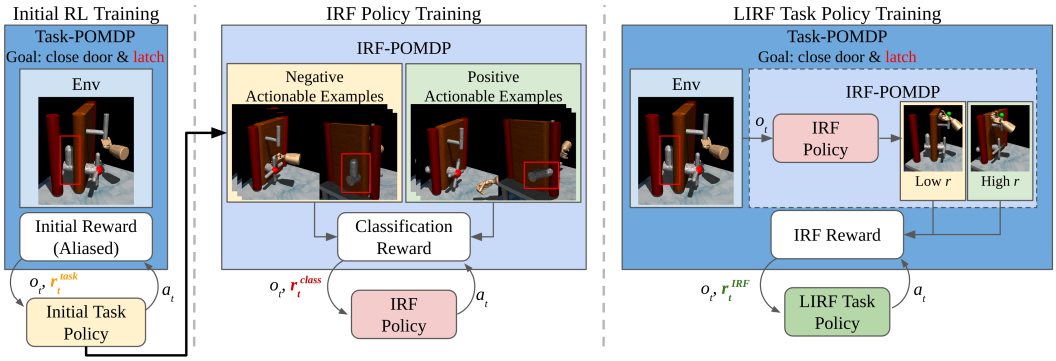

Figure 2: **Method Overview.** In a partially observable environment, (left) we first learn an initial task policy π_T^0 using passive classifier-based rewards. (middle) Then we train an IRF policy π_R to distinguish between provided "actionable positive examples" and π_T^0-generated negative examples. (right) Finally, we use π_R to provide the correct rewards for training a LIRF task policy π_T.

observability. This implies relying on additional sensors for reward estimation during training that are not available to the policy π_T that is being trained. In other words, while the states s_t are no longer observable to π_T, it may still be possible to learn $\hat{R}(s_t)$ through privileged instrumentation of the environment during training to allow the reward function to observe the full state s_t. For example, privileged thermal cameras could measure fluid levels to train an RGB image-based fluid pouring policy [13]. However, such additional instrumentation is labor-intensive, task-specific, and sacrifices the appealing scalability and ease of use of the exemplar rewards framework.

What is the alternative? Learning exemplar rewards $\hat{R}(o_t)$ based on the agent's instantaneous observations would suffer from state aliasing: in the door locking setting, such a reward function would not be able to distinguish a correctly latched door from one in which the latch is off by a factor of 90°! Nor does the exemplar reward learning framework permit relying on observation histories, like the solution we described above for POMDP *policies*. To see this, recall that exemplar rewards are trained from samples of successful task *outcomes* rather than full demonstrations, so there is no observation-action history information available for training $\hat{R}(o_{t-H:t}, a_{t-H-1:t-1})$.

3.1 Actionable Examples and Interactive Reward Functions (IRFs)

Our primary contribution is a solution to this conundrum: rather than presenting successful outcome examples as singular pre-recorded observations, we propose to present them as *"actionable examples"*, with which the robot can interact and generate new observations. In other words, to learn a reward function for locking a door, the robot gets to see several examples of successfully locked doors. Then, instead of training passive reward functions from image observations, we propose learning *"interactive reward functions"* (IRF), which consist of robot action policies π_R that reveal the task rewards. See the schematic in Figure 2. We expand upon this idea below.

First, we gather "actionable success examples" $P = \{\sigma_1^+, ..., \sigma_K^+\}$ and "actionable failure examples" $N = \{\sigma_1^-, ..., \sigma_K^-\}$. P and N contains object or environment configurations: in Figure 1, each σ^+ might be a tight bottle cap, a well-locked door, a stable tower, and so on. Now, the IRF policy may be seen as the solution to a new "IRF POMDP". This is identical to the task POMDP, except for two key differences: (1) It is initialized to an actionable example state $s_0 = \sigma \sim P \cup N$. (2) Its reward function is based on discovering the label of s_0: was it drawn from P or from N?[2]

Algorithm 1 shows pseudocode for our "Learning from IRFs" (LIRF) approach. We point out three key features here. **(1)** First, we bootstrap the task policy π_T by training against a single-observation-based reward classifier $D(o_t)$. This corresponds to running prior exemplar reward approaches; we use VICE [1]. **(2)** Next, in keeping with the exemplar rewards framework, we manually generate only the positive examples P. Negative examples N are instead generated by the above-initialized task policy. Thus, the IRF policy π_R serves as a GAN-like adversarial critic [18, 33] for training the

[2]In some experiments, we also sometimes use different actions for π_R than for π_T.

Algorithm 1 Learning from Interactive Reward Functions (LIRF) Framework

Require: a set P of positive actionable examples that specify successful task execution.

1: Following exemplar rewards approaches such as VICE [1], train a single-observation reward $D(o) \to [0, 1]$ and an initial task policy π_T^0 in the task POMDP.

2: Rollout π_T^0 for n times, collect negative actionable outcomes in N.

3: Train an IRF policy π_R in the IRF POMDP, where the environment is initialized from $s_0 = \sigma \sim P \cup N$, to maximize the reward:

$$\mathop{\mathbb{E}}_{s_0 \in P}\left[\sum_t \log(D(o_t))\right] + \mathop{\mathbb{E}}_{s_0 \in N}\left[\sum_t \log(1 - D(o_t))\right] \tag{1}$$

4: Train the final LIRF task policy π_T in the task POMDP, modified to include the additional terminal state reward $\hat{R}(o_T) = D(o_T) + \lambda D(o^*)$, where o_T is the observation for the terminal state s_T, and o^* is generated by the IRF policy π_R as $s_T \xrightarrow{\pi_R} s^*$, $o^* \sim O(s^*)$.

task policy π_T. **(3)** Finally, how could we discover the true task completion reward, i.e., the label P (corresponding to positive reward) or the label N (negative reward) at the end of IRF execution? For this, we could, in theory, train a classifier from the full IRF trajectory history. Instead, we find that it suffices to reuse the single-observation classifier $D(o_t)$ from (1) above, and apply it only to the final state after executing π_R. Intuitively, π_R learns to modify the environment state $\sigma \xrightarrow{\pi_R} s^*$ such that $o^* \sim O(s^*)$ reveals whether σ was drawn from P or N. For example, in door locking, π_R might learn to tug at the door: if the door is correctly locked (P), it stays closed, and if it is not locked (N), it opens. Thus, a single post-IRF observation suffices to classify σ. Figure 2 shows a schematic.

3.2 IRFs as Verification Mechanisms for Introspective, Fault-Tolerant Behaviors

Next, we observe that IRFs π_R rely on no privileged information beyond the task policy π_T's own observations. Thus, while their primary purpose is to provide reward functions for training task policies, IRFs can also be deployed together with the task policies after training. We propose a simple approach to use IRFs as verification mechanisms *in-the-loop* during task execution. Once the task policy π_T is executed for a fixed episode length, we will run the IRF π_R to determine whether the task is complete. If it is detected as incomplete, we will resume executing the task policy. This perform-verify cycle continues until the IRF is satisfied that the task is complete (or timeout). Refer to Appendix A.1 for the pseudocode.

4 Other Related Work

LIRF trains task policies through rewards computed by an interactive reward function, which is itself a robot policy trained from examples. While Section 2 extensively discussed connections to reward design, exemplar rewards, and POMDP policy-learning to set up our approach, we now position our contributions against two other key related areas of prior work.

Adversarial Robot RL. Like LIRF, some prior works train a target task policy against another "adversary" policy [34–37]. Pinto et al. [37] train a grasping robot to select more stable grasps by competing against an adversary robot that tugs at the grasped objects to dislodge them. Rather than adversary policies that destabilize the task policy to increase robustness, we train adversary policies that provide outcome-based rewards to train the task policy.

Interactive Perception and Verification. Our method draws on ideas from interactive perception [38], where agents in partially observed settings act to obtain information about the latent state. Interactive task verification mechanisms (see [39] for a survey) employ human-specified interactive perception behaviors to assess the state of a task. For example, a robot may use engineered motion primitives or perturbations to detect and correct task failures in manipulation [40, 41]. In contrast, we learn interactive verification behaviors from examples, and use them not only during task execution, but also to define the reward function for training task skill policies. One way to frame our contribution here is as follows: in reinforcement learning, agents emit actions into the environment

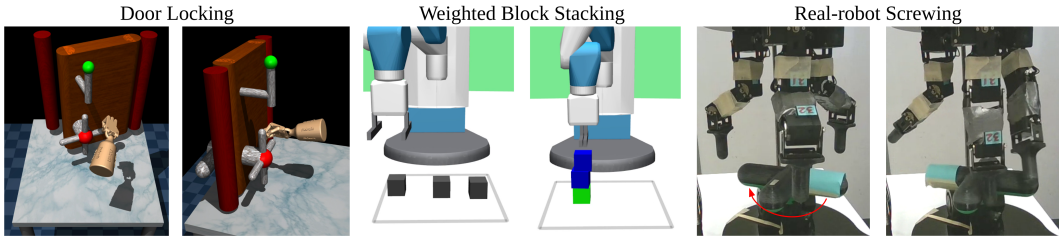

Figure 3: Our three tasks. For each pair of 2 images, we show the task initialization on the left, and the goal state on the right. In the middle panel, the 3 blocks are visually identical but they differ in weight. The heaviest block is green for visualization. In the right panel, we put a blue mark on the valve for visualizing the orientation.

and receive in return not just new states, but also *task rewards*. To our knowledge, LIRF is the first approach that employs interactive perception to directly estimate task rewards, rather than the state.

5 Experiments

Task Setups We evaluate LIRF on 3 tasks, illustrated in Figure 3, and described below. See Appendix A.2 for details and Supp slides for experiment videos.

• **Door Locking (Sim):** This is the door locking task used as a running example in sections 2.2 and 3. We instantiate the task with an Adroit Hand using MuJoCo [42]. The robot has a fixed viewpoint in front of the door, and must close the door and rotate the symmetric, four-handled doorknob to fully lock it (see Figure 3). The observation space mimics perfect vision: it consists of the door orientation, the door frame position, the *aliased* cross-handle orientation $[0, 90°)$[3] and the Adroit hand pose. The door is initialized as open, and the position of the door frame and the orientation of the latch are randomized during each initialization. The IRF policy should pull the door in order to verify the lock. We use open and close hand primitives for both task and IRF policies.

• **Weighted Block Stacking (Sim):** We further test our algorithm in a weighted block stacking environment based on MuJoCo Fetch [43] as shown in Figure 3. The goal of this task is to stack 3 visually identical blocks, initially in randomized positions, into a tower that is as stable and robust to perturbation as possible. One of the blocks (shown in green in figures for visualization purposes) is significantly heavier than the other two blocks, so the optimal strategy is to place it at the bottom. Again, the observation space mimics perfect vision: only the unordered poses of blocks are observable. The weight of a block is only revealed after it is picked up, to mimic physically plausible "hefting" behavior. The IRF policy must perturb the tower to evaluate its stability. We use pick-and-place primitives for the task policies, and poking primitives for the IRF policies to improve training efficiency.

• **Screw Tightening (Real):** To examine the robustness and generalizability of our algorithm, we test it on a real-robot screwing experiment using a D'Claw [44] that has 9 joints (Figure 3). The objective of this task is to turn the 4-prong valve clock-wise for around $180°$ into the "tightened" state (white line on the valve base). We engage a motor underneath the valve to mimic screw locking. Again, the observation consists of the historical aliased valve angles, i.e. $[0, 90°)$ and robot joint angles. IRF policies have to perturb the valve in order to determine whether the "screw" is locked.

LIRF Implementation Details. We use the soft actor-critic (SAC) algorithm [45] to train both the task policies and the IRF policies for each of the experiment and report results with mean and variance over 5 seeds for each simulated task. We heuristically set λ, the weight for the sparse IRF reward in Eq 1, to 1000 for all tasks. All policies are trained until convergence, which takes around 2M simulation steps for door locking and block stacking LIRF policies, 50k steps for screwing LIRF policies (~ 10 hours on the real robot), and around 20k steps for all IRF policies. We use MLPs with 3-5 layers for all policies, critics and reward functions. The networks are trained using the Adam optimizer with a learning rate of 3e-4 and a weight decay of 0. More details in Appendix A.5. We will release all code and environments.

[3] Due to symmetry, the cross-handle looks visually identical at 10°, as at 100 °, 190 °, and 280 °

	Task Specification	Door Locking	Block Stacking	Screwing
VICE [1]	Goal Examples	0.032 ± 0.015	0.310 ± 0.056	0.02
LIRF (Ours)	Actionable Goal Examples	$\mathbf{0.640 \pm 0.053}$	$\mathbf{0.846 \pm 0.027}$	0.82
LIRF+Verify (Ours)	Actionable Goal Examples	$\mathbf{0.958 \pm 0.007}$	$\mathbf{0.884 \pm 0.018}$	**0.99**
GAIfO [46]	Demonstrations	0.112 ± 0.024	0.275 ± 0.043	0.0
Manual IRF	Human Engineering	-	0.613 ± 0.058	0.85
GT State Reward	State+Human Engineering	0.714 ± 0.047	0.956 ± 0.021	0.87

Table 1: Task success rates of our method and baselines on the three tasks.

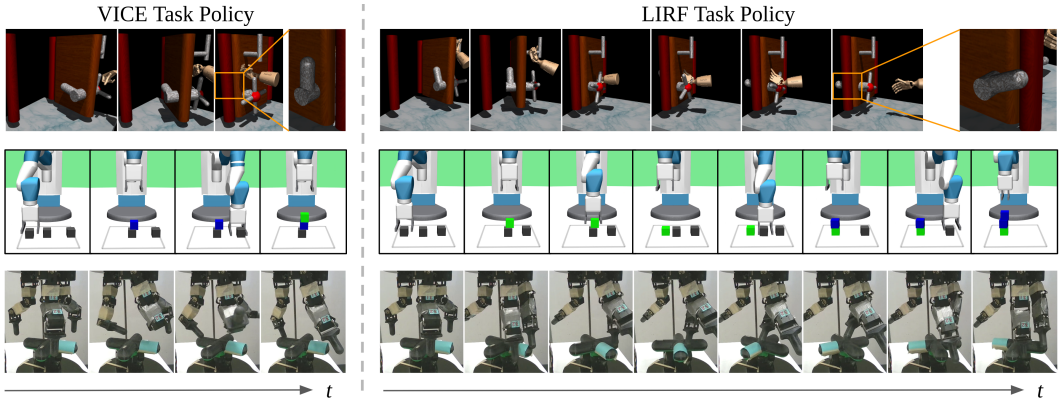

Figure 5: LIRF task policy rollouts (right), compared to VICE (left). Note row 2, where VICE builds an unstable tower with the heavy block on top (visualized in green here, but this is not observable to the agent). LIRF self-corrects mid-episode to build the tower with heavy block on bottom. See Supp slides for more video examples.

Baselines For evaluating our LIRF algorithm, we compare against RL policies learned with VICE exemplar rewards [1] and with ground truth state-based rewards ("GT State Reward"). In addition, we compare against GAIfO [46], an imitation-from-observation algorithm that learns from demonstrations. We provide as many demonstrations to GAIfO as the number of positive examples provided to LIRF (namely, 250, 10k, and 100 for door locking, weighted block stacking, and screwing respectively). Finally, when feasible, we engineer an interactive reward function policy (Manual IRF) that performs hand-coded actions to evaluate the task: hand-coded unscrewing behavior for screwing, and a random poking action for the block stacking.[4] See Appendix A.4 for more details on the baselines. Note that GAIfO, GT State Reward, and Manual IRF all involve more painstaking and comprehensive task specifications than LIRF, but offer useful comparison points nevertheless.

5.1 Results

Task Success Rates. Table 1 shows task success rates for our approach and baselines. As expected, VICE performs poorly due to state aliasing, but LIRF, which starts by training VICE policies (Algorithm 1), dramatically boosts this performance. As expected, GT State Reward, which uses a manually engineered reward based on the fully known state performs better than LIRF policies. Like VICE, GAIfO struggles to overcome state aliasing. On screwing, Manual IRF's hand-coded IRF is just a simple perturbation of the valve, as described in Appendix A.4, so it performs similarly to LIRF. On Weighted Block Stacking, Manual IRF is hard to hand-code, and ends up not working as well as our learned IRF policies, which is reflected in the task policy performance. LIRF+Verify, which reuses learned IRFs during task execution, achieves the best performance across the board, even beating GT State Reward on door locking and screwing.

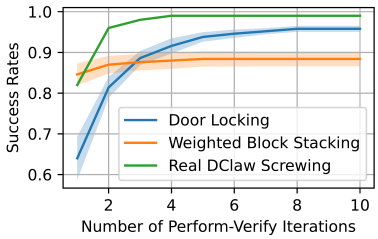

Figure 4: LIRF+Verify success rates vs. perform-verify iterations.

[4]For Adroit hand door locking, the action space is too large to reasonably hand-code any IRF behavior.

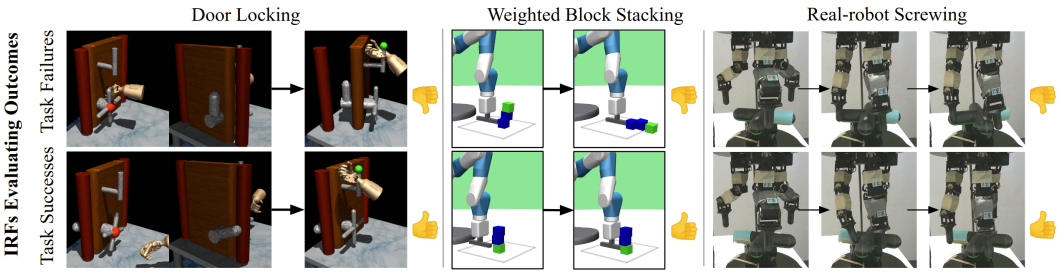

Figure 6: Running the IRF policy π_R on the success and failure actionable examples. Left (Door Locking): π_R grasps the door and pulls to verify locking. Middle (Block Stacking): π_R checks for tower stability by gently poking the bottom block. Right (Screwing): π_R attempts to lightly unscrew - the screw in the top row rotates as it is not tight, whereas the bottom-row does not.

What do the LIRF task policies π_T do? See Figure 5 for sample rollouts of LIRF policies, compared to VICE policies. For door locking, our task policy first closes the door then turns the cross-handle to lock the latch, as expected. For weighted block stacking, LIRF shows learned self-correction behavior (even without LIRF+Verify): in episodes where the heavier block is incorrectly stacked on the lighter block, LIRF rebuilds the tower in the correct order 87.3% percent of the time (see Figure 5). For screwing, LIRF turns the screw clockwise to tighten. Figure 4 shows how LIRF+Verify improves performance with more perform-verify iterations. LIRF+Verify makes the biggest difference in door locking: here, plain LIRF fails mainly because the policies operating on partial observations do not learn precisely how much to turn the latch without the aid of π_R to declare task completion. This is also why GT State Reward performs quite poorly on this task despite training on the true state-based reward function. LIRF+Verify's gains over LIRF are smaller in block stacking and screwing, because LIRF failures here occur from low-level imprecision in pick-and-place or valve-turning. Here, LIRF+Verify helps by allowing the task policy π_T to try again when it fails. We show video examples in Supp slides.

What do the IRF policies π_R do? Figure 6 shows examples of learned reward function policies. First, IRFs learn sensible strategies to separate visually indistinguishable positives and negatives. In the door locking task, IRF pulls the fixed handle; in stacking, it pokes the bottom block; in screwing, it applies a small counterclockwise torque to unscrew the valve. Next, as Table 2 shows, IRF-based task success classification is highly accurate for all tasks. Finally, IRF policy learning is quite efficient in terms of the number of actionable positive examples required. Door locking, screwing, and weighted block stacking need 250, 100, and 10k examples respectively. Block stacking is hardest, since distinguishing stable from unstable towers requires a finely tuned poking force and motion. See Appendix A.6 for more details.

Task	Accuracy
Door Locking	0.992 ± 0.006
Block Stacking	0.852 ± 0.023
Screwing	0.98

Table 2: IRF task success classification accuracies.

6 Conclusions

We have presented LIRF, a framework for conveniently training example-based interactive robot policies to evaluate robot task policies, in order both to provide rewards to train them, and to verify their execution in partially observed tasks. While our results show substantial improvements over baselines, LIRF currently has two drawbacks. Our method trains IRF evaluation policies under the assumption that the initial task policy outcomes are mostly failures, and might struggle to learn good IRF behaviors when this is not the case, such as when single-image-based rewards produce effective policies, or if we extend the LIRF algorithm to have multiple adversarial updates of task and IRF policies. Next, LIRF requires the physical storage of "actionable outcomes" as positive examples. While the number of such examples in some tasks may be small enough to not be a major concern (e.g., among our experiments, 100 for screwing, 250 for door locking), in other cases, storing and presenting a large number of positive example objects to train the IRF might be impractical. Future work overcoming these limitations would be make our approach more broadly applicable.

References

[1] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational inverse control with events: A general framework for data-driven reward definition. *Advances in neural information processing systems*, 31, 2018.

[2] A. Xie, A. Singh, S. Levine, and C. Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, pages 40–52. PMLR, 2018.

[3] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[4] B. Eysenbach, S. Levine, and R. R. Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] K. Konyushkova, K. Zolna, Y. Aytar, A. Novikov, S. Reed, S. Cabi, and N. de Freitas. Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*, 2020.

[6] K. Zolna, A. Novikov, K. Konyushkova, C. Gulcehre, Z. Wang, Y. Aytar, M. Denil, N. de Freitas, and S. Reed. Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*, 2020.

[7] Y. J. Ma, A. Shen, D. Jayaraman, and O. Bastani. Smodice: Versatile offline imitation learning via state occupancy matching. *arXiv preprint arXiv:2202.02433*, 2022.

[8] S. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

[9] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 59–66. IEEE, 2019.

[10] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100. PMLR, 2022.

[11] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[12] OpenAI. Gym: A toolkit for developing and comparing reinforcement learning algorithms. https://github.com/openai/gym.

[13] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2629–2636. IEEE, 2017.

[14] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86. IEEE, 2017.

[15] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[16] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[17] S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[18] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

[19] W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16, 2009.

[20] V. Veeriah, P. M. Pilarski, and R. S. Sutton. Face valuing: Training user interfaces with facial expressions and reinforcement learning. *arXiv preprint arXiv:1606.02807*, 2016.

[21] S. Chernova and A. L. Thomaz. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121, 2014.

[22] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[23] J. Lin, Z. Ma, R. Gomez, K. Nakamura, B. He, and G. Li. A review on interactive reinforcement learning from human social feedback. *IEEE Access*, 8:120757–120765, 2020.

[24] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[25] R. A. McCallum. Overcoming incomplete perception with utile distinction memory. 1993.

[26] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[27] P. Zhu, X. Li, P. Poupart, and G. Miao. On improving deep reinforcement learning for pomdps. *arXiv preprint arXiv:1704.07978*, 2017.

[28] T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent model-free rl is a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

[29] K. Gregor, D. Jimenez Rezende, F. Besse, Y. Wu, H. Merzic, and A. van den Oord. Shaping belief states with generative environment models for rl. *Advances in Neural Information Processing Systems*, 32:13475–13487, 2019.

[30] T. Gangwani, J. Lehman, Q. Liu, and J. Peng. Learning belief representations for imitation learning in pomdps. In *Uncertainty in Artificial Intelligence*, pages 1061–1071. PMLR, 2020.

[31] G. Weisz, P. Budzianowski, P.-H. Su, and M. Gašić. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(11):2083–2097, 2018.

[32] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for POMDPs. June 2018.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[34] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR, 2017.

[35] X. Pan, D. Seita, Y. Gao, and J. Canny. Risk averse robust adversarial reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8522–8528. IEEE, 2019.

[36] X. Ma, K. Driggs-Campbell, and M. J. Kochenderfer. Improved robustness and safety for autonomous vehicle control with adversarial reinforcement learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1665–1671. IEEE, 2018.

[37] L. Pinto, J. Davidson, and A. Gupta. Supervision via competition: Robot adversaries for learning tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1601–1608. IEEE, 2017.

[38] J. Bohg, K. Hausman, B. Sankaran, O. Brock, D. Kragic, S. Schaal, and G. S. Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

[39] O. Kroemer, S. Niekum, and G. D. Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research*, 22(30), 2021.

[40] Z. Su, O. Kroemer, G. E. Loeb, G. S. Sukhatme, and S. Schaal. Learning manipulation graphs from demonstrations using multimodal sensory signals. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2758–2765. IEEE, 2018.

[41] S. Niekum, S. Osentoski, C. G. Atkeson, and A. G. Barto. Online bayesian changepoint detection for articulated motion models. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1468–1475. IEEE, 2015.

[42] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[43] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[44] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on robot learning*, pages 1300–1313. PMLR, 2020.

[45] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[46] F. Torabi, G. Warnell, and P. Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.