Noname manuscript No.
(will be inserted by the editor)

Active Reward Learning with a Novel Acquisition Function

Christian Daniel · Oliver Kroemer · Malte Viering · Jan Metz · Jan Peters

Received: date / Accepted: date

Abstract Reward functions are an essential component of many robot learning methods. Defining such functions, however, remains hard in many practical applications. For tasks such as grasping, there are no reliable success measures available. Defining reward functions by hand requires extensive task knowledge and often leads to undesired emergent behavior. We introduce a framework, wherein the robot simultaneously learns an action policy and a model of the reward function by actively querying a human expert for ratings. We represent the reward model using a Gaussian process and evaluate several classical acquisition functions from the Bayesian optimization literature in this context. Furthermore, we present a novel acquisition function, expected policy divergence. We demonstrate results of our method for a robot grasping task and show that the learned reward function generalizes to a similar task. Additionally, we evaluate the proposed novel acquisition function on a real robot pendulum swing-up task.

Christian Daniel · Malte Viering ·
Jan Metz · Oliver Kroemer · Jan Peters
Technische Universität Darmstadt
Hochschulstrasse 10
64289 Darmstadt
Germany
Tel.: +49-6151-166167
Fax: +49-6151-167374
E-mail: {*lastname*}@ias.tu-darmstadt.de

Jan Peters
Spemannstraße 38
Max-Planck-Institut für Intelligente Systeme
72076 Tübingen
Germany

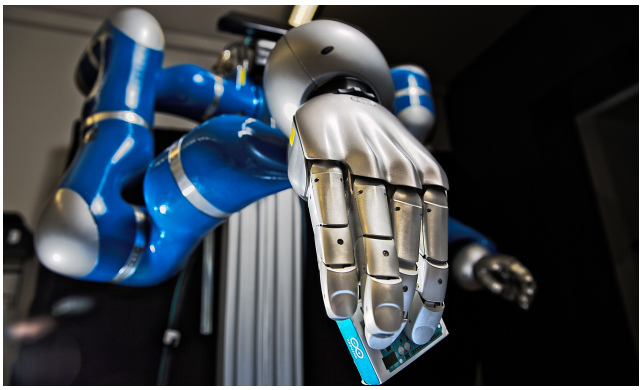

Fig. 1: The Robot-Grasping Task. While grasping is one of the most researched robotic tasks, finding a good reward function still proves difficult.

Keywords Reinforcement Learning · Active Learning · Bayesian Optimization · Preference Learning · Inverse Reinforcement Learning · Reward Functions · Acquisition Functions

1 Introduction

An important goal of Reinforcement Learning (RL) is to yield more autonomous robots. However, RL methods require reward functions to specify desired behavior. While such reward functions are usually hand coded, defining them for real robot tasks is challenging even for well-studied problems such as grasping. Thus, hand coding reward functions is shifting the problem of requiring an expert to design a hard-coded controller to requiring a hard-coded reward function.

Despite the variety of grasp stability measures that have been developed [38], it has been shown that the resulting grasps are outperformed by grasps learned from kinesthetic teach-in [3]. This example shows that even experts will often design reward functions that are not effective in practice, or lead to undesired emergent behavior [30], while even non-experts can indicate good grasps. While it may be difficult to analytically design a reward function or to give demonstrations, it is often easy for an expert to rate an agent's executions of a task. Thus, a promising alternative is to use the human not as an expert in performing the task, but as an expert in evaluating task executions. Unfortunately, as already shown by Thomaz and Breazeal [39] and by Cakmak and Thomaz [5], humans have considerable noise in their ratings of actions. To deal with imprecise human ratings, we propose to learn a probabilistic model of the reward function and use it to guide the learning process of a RL agent. However, while learning a complicated task will typically require many rollouts, a good estimate of the reward model can often by inferred from less samples. Thus, our goal is to learn a model of

the human's intrinsic reward function that generalizes to similar observations such that the agent can learn the task from few human-robot interactions.

To avoid specifying reward functions, Inverse RL (IRL) extracts a reward function from demonstrations [33, 34, 41]. For many tasks, such demonstrations can be obtained by kinesthetic teach-in or teleoperation. Unfortunately some skills – such as throwing a basket ball – are hard to demonstrate. Moreover, both methods require sufficient proficiency of the demonstrator which requires being good at a task and being good at performing the task on a robot. Following this intuition, preference based algorithms allow the expert to rank executions and learn a controller based on these rankings [2, 6]. The commonly used approach in ranking is to let the expert rank the previously best sample against the current sample. This approach presents an intriguing idea, as humans are better at giving relative judgments than absolute judgments. However, this approach only provides a single bit of information. The technical term describing how much information human subjects can transmit for a given input stimuli is called channel capacity [27]. The *channel capacity* is a measure for how many input stimuli subjects can distinguish between. In a review of several experiments, Miller [27] concludes that humans have a general channel capacity between 1.6 and 3.9 bits for unidimensional stimuli. Adding more dimensions to the stimuli can further increase this value. Experiments have also been performed to find out whether humans can transmit more information when labeling stimuli according to predescribed categories or when being allowed to rate on a scale [15]. While there was no significant difference, subjects performed better when rating on a scale. Based on these insights, we design our framework such that the human expert can assign numerical values to observed demonstrations. Furthermore, numerical rewards allow the human to indicate strong preferences over demonstrations.

In this paper, we take advantage of the Gaussian Process (GP) regression framework to represent the reward model as well as the Bayesian Optimization (BO) literature to optimize this model. We show that we can use standard BO methods to efficiently minimize the number of expert interactions, which is essential when designing methods for 'autonomous' agents. Furthermore, we introduce a novel acquisition function (AF) that outperforms traditional AFs in the proposed setting. We evaluate the proposed method on a series of simulated and real robot tasks. We also show that a reward function which is learned for one task (grasping a box) can be reused to learn a similar task (grasping a pestle).

2 Reward Learning Framework

In this paper, we present an approach that complements the Reinforcement Learning (RL) framework by replacing the hard-coded reward function by a learned one. As most RL methods do not make assumptions about the reward function, the proposed approach does not replace existing RL methods but can be used to extend them. We base our discussion on the Policy Search (PS) framework. While there are many approaches to RL, PS is especially well suited for real robot tasks. It does not rely on exhaustive sampling of the state-action space and, as a result, many recent advances in robot learning are based on PS [23, 21, 29]. We consider the contextual episodic PS case with continuous contexts $\boldsymbol{s} \in \mathcal{S}$ and continuous control parameters $\boldsymbol{\omega} \in \Omega$. We use the term 'context' to describe the subset of information contained in the initial state that is necessary to describe the task at hand. Examples could be the position of an object or desired target locations. The distribution over contexts is denoted by $\mu^\pi(\cdot)$. Whereas the traditional RL notation uses actions $\boldsymbol{a}$, we instead write control parameters $\boldsymbol{\omega}$ to clarify that we directly optimize for the parameters of a controller which can, for example, be a Movement Primitive as discussed in Section 5.2. Executing a rollout with control parameters $\boldsymbol{\omega}$ in context $\boldsymbol{s}$ results in a trajectory $\boldsymbol{\tau} \sim p(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\omega})$, where the trajectory encodes both, the robot's internal transitions as well as relevant environment transitions (e.g. joint angles and a ball's position and speed). The agent starts with an initial control policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$ in iteration 0 and performs a predetermined number of rollouts. After each iteration, the agent updates its policy to maximize the expected reward

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau}}[R(\boldsymbol{\tau})] = \iiint R(\boldsymbol{\tau})\, p(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{\omega}, \tag{1}$$

with $p(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{\tau}) = p(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\omega})\pi(\boldsymbol{\omega}|\boldsymbol{s})\mu^\pi(\boldsymbol{s})$. Parameter-based PS has been shown to be well suited for complex robot tasks [9], as it abstracts complexity while retaining sufficient flexibility in combination with suitable movement primitive representations.

In the original RL setting, a reward function for evaluating the agent's behavior is assumed to be known. While not always explicitly stated, the reward function for real robot tasks often depends on features $\phi(\boldsymbol{\tau})$ of the trajectory. We refer to the result of the feature extraction from trajectories as the *outcomes* $\boldsymbol{o} = \phi(\boldsymbol{\tau})$. We assume that the reward depends only on the features of the trajectories. Such outcomes could, for example, describe the minimal distance to goal positions or the accumulative energy consumption.

As the problem of specifying good features $\phi(\boldsymbol{\tau})$ is beyond the scope of this paper, we assume the features

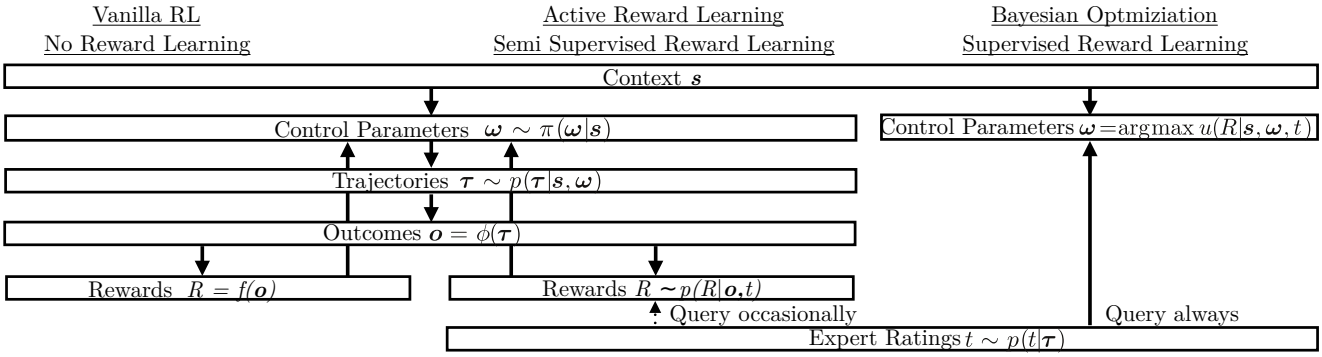

Fig. 2: Sketch of the elements of different possible approaches. The left column shows the 'vanilla' RL where a reward function is assumed. The middle column shows the proposed active reward learning approach, which shares the policy learning component with vanilla RL but models a probabilistic reward model that gets updated by asking for expert ratings sometimes. The right column shows the BO approach, where actions are chosen that maximize the utility function (for example one of the acquisition functions presented in 3 and requires an expert rating for each of the chosen actions).

to be known, as is usually assumed when designing reward functions. The results in Section 6 show that the proposed method learns complicated tasks using simple features. This is possible because we base the estimation of our probabilistic reward model on kernel functions with automatic relevance determination, such that the importance of individual features can be estimated from data [32].

As outcomes are of much lower dimensionality than trajectories, we can use outcomes to efficiently model the reward function $\hat{R}(\boldsymbol{o})$ from a training set $\mathcal{D} = \{\boldsymbol{o}_{1:n}, \boldsymbol{R}_{1:n}\}$ using regression. Using the feature representation, the term $R(\boldsymbol{\tau})$ in Eq. (1) is replaced by $R\left(\boldsymbol{o}{=}\phi(\boldsymbol{\tau})\right)$ to become

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau}}[R(\boldsymbol{o})] = \iiint R\left(\boldsymbol{o}{=}\phi(\boldsymbol{\tau})\right) p(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{\tau}) \, \mathrm{d}\boldsymbol{\tau} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega}. \tag{2}$$

Obviously, it is not sufficient to build the reward function model solely on samples from the initial policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$, as the agent is likely to exhibit poor performance in the early stages of learning and the reward learner would not observe good samples. Therefore, we need to couple the process of learning a good policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with learning a good reward function $\hat{R}(\boldsymbol{o})$, such that they are developed interdependently. However, in such a coupled active learning process, we need to know which samples to query from our expert to minimize expert interaction.

Modeling a probabilistic reward model $p(R|\boldsymbol{o}, \mathcal{D})$, instead of a deterministic reward function $R(\boldsymbol{o})$, allows us to leverage the information about the certainty of our estimate to control the amount of human interaction required, as we only need to interact with the human expert if our model of the reward is not sufficiently certain. We describe the details of this process in Section 2.1.

When using a probabilistic model of the reward, we have to replace the reward term in Eq. (1) with

$$R(\boldsymbol{o}) = \mathbb{E}_{p(R|\boldsymbol{o})}[R] = \int p(R|\boldsymbol{o}) \; R \, \mathrm{d}R,$$

as most PS methods work on the expectation of the reward. Using the probabilistic model of the reward function, the PS method continuously learns how to achieve high reward outcomes, while the reward model learner adapts the accuracy of its model.

2.1 Active Reward Learning

The proposed approach is independent of and complementary to the RL method used. The RL method aims to find the optimal policy $\pi^*(\boldsymbol{\omega}|\boldsymbol{s})$ and is agnostic as to how the rewards are computed. Similarly, Active Reward Learning (ARL) is agnostic as to how the set of available samples are created. Hence, the method for actively learning the reward can be combined with any RL method. It follows that while RL methods are inherently active, and often aim to represent a reward-related function such as the value function internally, the proposed approach solves a distinct learning problem and has to be categorized separately. ARL aims to maximize its relevant information about the human's intrinsic reward model in a sample efficient manner. Thus, rather than annotating all samples, we aim to find a method that actively selects informative samples to query. Hence, learning the human's intrinsic reward function is an active learning process.

Our goal is to find a model $p(R|\boldsymbol{o}, \mathcal{D})$ that predicts the reward R given an observed outcome $\boldsymbol{o}$ and training data $\mathcal{D}$, obtained from an expert. When modeling the

Input: Information loss tolerance ϵ, improvement threshold λ, acquisition function u

Initialize π using a single Gaussian with random mean. GP with zero mean prior.

while not converged

- **Set sample policy:** $q(\boldsymbol{\omega}|\boldsymbol{s}) = \pi_{\text{old}}(\boldsymbol{\omega}|\boldsymbol{s})$
- **Sample:** collect samples from the sample policy $\{\boldsymbol{s}_i \sim p(\boldsymbol{s}), \boldsymbol{\omega}_i \sim q(\boldsymbol{\omega}|\boldsymbol{s}_i), \boldsymbol{o}_i\}, i \in \{1, \dots, N\}$
- **Define Bellman Error Function** $\delta(\boldsymbol{s}, \boldsymbol{\omega}) = R(\boldsymbol{o}) - V(\boldsymbol{s})$
- **Minimize the dual function** $[\boldsymbol{\alpha}^*, \eta^*] = \arg\min_{[\boldsymbol{\alpha}, \eta]} g(\boldsymbol{\alpha}, \eta)$
- **Determine base line** $V(\boldsymbol{s}) = \boldsymbol{\alpha}^{T*} \boldsymbol{\phi}(\boldsymbol{s})$
- **Policy update:**
  Calculate weighting $p(\boldsymbol{s}_i, \boldsymbol{\omega}_i) \propto q(\boldsymbol{s}_i, \boldsymbol{\omega}_i) \exp\left(\frac{1}{\eta^*} \delta^*(\boldsymbol{s}_i, \boldsymbol{\omega}_i)\right)$
  Estimate distribution $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ by weighted maximum likelihood estimates
- **Reward model update:**
  FindNominee = true
  while FindNominee
  - **Nominate outcome:** $\boldsymbol{o}^+ = \arg\max u(\boldsymbol{o})$
    if $\left(\boldsymbol{o}^+ \notin \mathcal{D}\right) \wedge \left(\sigma(\boldsymbol{o}^+)/\beta > \lambda\right)$
    Demonstrate corresponding trajectory $\boldsymbol{\tau}^+$
    Query Expert Reward R^+
    $\mathcal{D} = D \cup \{\boldsymbol{o}^+, R^+\}$
    else
    FindNominee = false
  - Update reward model $p(R|\boldsymbol{o}, \mathcal{D})$
  - Optimize GP-hyper parameters $\boldsymbol{\theta}$

Output: Policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, reward model $p(R|\boldsymbol{o}, \mathcal{D})$

Table 1: We show the algorithmic form of active reward learning with REPS. We specify the information loss tolerance ϵ as well as an initial sampling policy and an improvement threshold λ. In each iteration, the algorithm first samples from the sampling policy, minimizes the dual function to find values for $\boldsymbol{\alpha}^{T*}$ and η^* and then computes the next policy. After each policy search iteration, the reward function learner chooses whether to demonstrate samples to the expert according to the acquisition function. The parameters $\boldsymbol{\alpha}$ and η are parameters of the dual function problem of REPS and can be optimized through standard optimization algorithms [9].

reward, we have to take into account that the expert can only give noisy samples of their implicit reward function and we also have to model this observation noise. Thus, we need to solve the regression problem

$$\tilde{R}(\boldsymbol{o}) = R(\boldsymbol{o}) + \eta, \ \eta \sim \mathcal{N}(0, \beta),$$

where we assume zero mean Gaussian noise. Such a regression problem can, for example, be solved with Gaussian Process (GP) regression

$$R(\boldsymbol{o}) \sim \mathcal{GP}(m(\boldsymbol{o}), k(\boldsymbol{o}, \boldsymbol{o}')),$$

where $m(\boldsymbol{o})$ is the mean function and $k(\boldsymbol{o}, \boldsymbol{o}')$ is the covariance function of the GP. For the remainder of this paper, we use the standard squared exponential covariance function

$$k(\boldsymbol{o}, \boldsymbol{o}') = \boldsymbol{\theta}_0^2 \exp\left(-\frac{||\boldsymbol{o} - \boldsymbol{o}'||^2}{2\boldsymbol{\theta}_1^2}\right).$$

The hyper parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \beta\}$ are found through optimization of the model's log likelihood [32]. Given a training set $\mathcal{D} = \{\boldsymbol{o}_{1:n}, \boldsymbol{R}_{1:n}\}$, we can write down the covariance matrix between previously observed rewards and outcomes

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{o}_1, \boldsymbol{o}_1) & \dots & k(\boldsymbol{o}_1, \boldsymbol{o}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{o}_n, \boldsymbol{o}_1) & \dots & k(\boldsymbol{o}_n, \boldsymbol{o}_n) \end{bmatrix} + \beta \boldsymbol{I}.$$

Assuming a mean zero prior, the joint Gaussian probability of the training samples in $\mathcal{D}$ and the reward prediction R^+ of a new unrated observation is given by

$$\begin{bmatrix} \boldsymbol{R}_{1:n} \\ R^+ \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} & \hat{\boldsymbol{k}} \\ \bar{\boldsymbol{k}} & k(\boldsymbol{o}^+, \boldsymbol{o}^+) \end{bmatrix}\right),$$

with $\hat{\boldsymbol{k}} = [k(\boldsymbol{o}_1, \boldsymbol{o}^+) \dots k(\boldsymbol{o}_n, \boldsymbol{o}^+)]^T$ and $\bar{\boldsymbol{k}} = [k(\boldsymbol{o}^+, \boldsymbol{o}_1) \dots k(\boldsymbol{o}^+, \boldsymbol{o}_n)]$. The predictive posterior reward $p(R^+|\boldsymbol{o}, \mathcal{D})$ of a new outcome $\boldsymbol{o}^+$ is then given by a Gaussian

$$p(R^+|\boldsymbol{o}, \mathcal{D}) \sim \mathcal{N}\left(\mu(\boldsymbol{o}^+), \sigma^2(\boldsymbol{o}^+)\right),$$

with mean and variance

$$\mu(\boldsymbol{o}^+) = \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{R}_{1:n},$$
$$\sigma^2(\boldsymbol{o}^+) = k(\boldsymbol{o}^+, \boldsymbol{o}^+) - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k} + \beta,$$

by conditioning the GP on the observed outcome $\boldsymbol{o}^+$. Using the GP, we can represent both our expected reward $\mu(\boldsymbol{o}^+)$, which is provided to the policy learner, and the variance of the reward $\sigma^2(\boldsymbol{o}^+)$ which is essential to the reward learner. The reward variance $\sigma^2(\boldsymbol{o}^+)$ depends on the distance of the outcome $\boldsymbol{o}^+$ to all outcomes in the training set $\mathcal{D}$ and the observation noise β, which is a hyper parameter that we optimize for. Using the predictive variance, we can employ one of many readily available BO methods to find the maximum of the reward function.

3 Bayesian Optimization for Active Reward Learning

In this section, we first describe how to adapt arbitrary acquisition to the presented framework. Subsequently, we present a novel acquisition function which is optimized for active reward learning. The goal of BO

is to optimize a function under uncertainty. Acquisition functions (AFs) are utility functions whose function value is maximized at locations of the input space which are likely to be maxima of the original problem. AFs usually encode an exploration-exploitation trade-off, such that they not only query samples in known high value regions but also in regions that have not been sufficiently explored before. While using GPs to model the reward function allows us to employ BO, we deviate from the general BO approach in two points.

First, as our GP models a relationship of outcomes to rewards instead of context-actions to rewards, we cannot sample arbitrary outcomes $\hat{\boldsymbol{o}}$ to improve our estimate, as the robot does not know how to produce these actions. To do so, we would require access to $p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$ such that we can request the agent to perform actions that result in trajectories $\hat{\boldsymbol{\tau}}$ which yield the outcome $\hat{\boldsymbol{o}} = \phi(\hat{\boldsymbol{\tau}})$. Additionally, we would need to guarantee that the outcomes requested by the AF are physically possible. However, this transition model is unknown and, thus, we are restricted to the set of previously observed outcomes that have been generated during the agent's learning process so far. Alternatively we could aim to learn $p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$. However, learning the transition model would usually require far more samples in the considered setting. Second, we need to balance the improvement of our current estimate of the reward function and the number of queries that we request from the expert, i.e., we want to find a trade-off between finding a good reward function and learning the task with minimal human input.

While restricting the set of outcomes that can be queried to previously generated outcomes is straightforward, balancing the number of queries requires additional constraints in the case of arbitrary AFs. In the following section we will introduce several techniques to optimize this trade-off.

3.1 Sample Efficiency

The traditional BO framework aims to find a global maximum. Sampling of the objective function can be terminated when the improvement of the function value is marginal, for example when the predicted variance around the optimum is very low. In the proposed scenario where a policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ and a reward model $p(R|\boldsymbol{o})$ are learned simultaneously and obtaining training samples of the reward model is very expensive, the problem of deciding *when* to improve the reward model becomes crucial.

The reward model relies on the policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ to provide outcomes in interesting, i.e., high reward regions, and the policy relies on the reward model $p(R|\boldsymbol{o})$ to guide its exploration towards such regions of interest. Thus, the reward model needs sufficient training data to approximately predict the reward function in early stages and a higher density of training points once the agents policy starts to converge to a solution. At the same time, we want to minimize the number of queries to the expert over the learning process.

In order to balance this trade-off, we propose an acquisition algorithm which, in accordance with the selected acquisition function $u(\boldsymbol{o})$, first samples the best observed sample outcome from the history of the agent's outcomes $\boldsymbol{o}_{n+1} = \arg\max_{\boldsymbol{o}} u(\boldsymbol{o}|\mathcal{D})$. In this sample based search, we additionally include all outcomes that have been rated by the expert. If the acquisition function is maximized by a previously rated outcome, we stop and do not query any samples in this iteration.

Maximizing the AF by an already observed outcome is unique to the sample based case. As for the continuous case, a point around an observed outcome would usually have higher variance and, thus, maximize the AF. With our approach, we achieve a sparse sampling behavior that requires fewer expert interactions. If, however, the outcome that maximizes u has not yet been rated by the expert, we need to decide whether querying the outcome is beneficial.

For example, we may have already converged to a good estimate of the reward function, and new outcomes improve on the mean reward solely due to the observation noise. In this case we do not want to query the expert. Thus, we decide whether to query an outcome by thresholding the ratio of predictive variance and estimated observation noise $\sigma(\boldsymbol{o})/\sqrt{\beta} > \lambda$, where λ is a tuning parameter which allows us to trade off the accuracy of the final solution with the query frequency by adapting the available AFs to explicitly take the estimated observation noise into account. While this adaptation of the AFs introduces a new parameter to be tuned, our experiments show that the use of this technique results in fewer human interactions while maintaining a high performance and tuning of the parameter becomes straightforward.

If the robot decides to query the sample's reward value, it updates the GP and searches for the new maximum of the acquisition function. Otherwise it stops and does not update the GP further in this iteration. Additionally, we can limit the history of samples the AF can access. Limiting the history can be useful when using local search methods (e.g., policy search methods). When outcomes that were produced earlier in the learning process no longer have relevance under the current policy (the policy does not have probability mass in these regions). Adding information about these regions cannot affect the learning process. Since general AFs

are not designed to evaluate the effect on the resulting policy they might propose samples that are informative for the reward model but not the policy learner.

4 Expected Policy Divergence

The tuning strategies (sampling threshold, sparse sampling, limiting the history of samples and not requerying known samples) detailed above allow us to use arbitrary acquisition functions in our framework. In order to follow a more principled approach without these heuristics, we introduce a novel acquisition function which is specialized to our use case. The proposed AF, Expected Policy Divergence (EPD) is based on the insight that, as opposed to the traditional BO case, we are not mainly interested in the modeled reward function itself, but rather in the policy which the robot learns through the reward model. Thus, we should not evaluate the improvement in the reward model but instead quantify the potential change of the policy given additional information in the form of expert ratings.

Generally, starting from the current policy $\pi(\boldsymbol{\omega})$ the robot can update this policy under the current reward model $p(R|\boldsymbol{o}, \mathcal{D})$ to get a new policy $\tilde{\pi}(\boldsymbol{\omega})$. Alternatively, the robot can first update its reward model to become $p^*(R|\boldsymbol{o}, \mathcal{D}^*)$ and then update the policy to get the policy $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ under the updated reward model. In this setting the difference between $\pi(\boldsymbol{\omega})$ and any successive policy (e.g., $\tilde{\pi}(\boldsymbol{\omega})$ or $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$) will often be controlled through some form of step size. However, the difference between $\tilde{\pi}(\boldsymbol{\omega})$ and $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ is only explained by the additional information gained by updating the reward model. In EPD, we aim to maximize the difference between $\tilde{\pi}(\boldsymbol{\omega})$ and $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$. Thus, if an additional sample would improve the reward model, but not affect the policy update, the human will not be queried.

Similarly to traditional AFs, we evaluate the set of possible sample locations $\boldsymbol{o} \in \mathcal{O}$ to find the most informative sample location. Given a sample $\boldsymbol{o}$, we assign a reward R according to our current reward model $R \sim p(R|\boldsymbol{o}, \mathcal{D})$ and add this pair to our data set $D^* = \mathcal{D} \cup \{(\boldsymbol{o}, R)\}$. This additional information will affect the reward model to become $p^*(R|\boldsymbol{o}, \mathcal{D}^*)$, such that the expected reward of all other outcomes is changed. Hence, we have to use the new reward model $p^*(R|\boldsymbol{o}, \mathcal{D}^*)$ to evaluate the policy update

$$\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) = f(\pi(\boldsymbol{\omega}), p^*(R|\boldsymbol{o}, \mathcal{D}^*)),$$

where $f(\cdot)$ is any policy update function and $\pi(\boldsymbol{\omega})$ is the stochastic policy.

Finally, we can compare the new policy $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ to the baseline policy $\tilde{\pi}(\boldsymbol{\omega})$ to quantify how much querying $\boldsymbol{o}$ influenced the policy update. The KL divergence,

$$\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \;||\; \tilde{\pi}(\boldsymbol{\omega})\right) = \int \pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \log \frac{\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)}{\tilde{\pi}(\boldsymbol{\omega})} \mathrm{d}\boldsymbol{\omega},$$

is a natural choice to quantify the difference between two policies. When working with a deterministic policy, the KL can be computed on the policy's mean. The proposed AF maximizes the expected KL divergence between the new policy $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ and the baseline $\tilde{\pi}(\boldsymbol{\omega})$,

$$\begin{aligned} \mathrm{EPD}(\boldsymbol{o}) &= \mathbb{E}_{p(R|\boldsymbol{o},\mathcal{D})}\left[\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \;||\; \tilde{\pi}(\boldsymbol{\omega})\right)\right] \\ &= \iint \pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \log \frac{\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)}{\tilde{\pi}(\boldsymbol{\omega})} p(R|\boldsymbol{o}, \mathcal{D}) \mathrm{d}\boldsymbol{\omega} \mathrm{d}R, \end{aligned} \tag{3}$$

where the baseline $\tilde{\pi}(\boldsymbol{\omega})$ is defined to be the policy obtained through the policy update $\tilde{\pi}(\boldsymbol{\omega}) = f(\pi(\boldsymbol{\omega}), p(R|\boldsymbol{o}, \mathcal{D}))$. Similar to the the sampling trade-off variable λ that we introduced for the traditional AFs, EPD also needs a sampling threshold. EPD will query a sample if

$$\mathbb{E}_{p(R|\boldsymbol{o},\mathcal{D})}\left[\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \;||\; \tilde{\pi}(\boldsymbol{\omega})\right)\right] \geq \kappa, \tag{4}$$

where κ is user specified. However, EPD does not rely on the additional tuning strategies used for generic AFs.

4.1 Practical Considerations

After presenting the theoretical foundations of EPD, we will now investigate some of the implementation details.

Expected Policy. Unfortunately, the EPD depends on the possibly nonlinear policy update and cannot generally be solved in closed form. Instead, we have to resort to sample-based methods. However, both the update of the reward model $p(R|\boldsymbol{o}, \mathcal{D})$ as well as the policy update are expensive operations and Monte-Carlo sampling should be avoided. Instead, we propose alternatives that are evaluated in the experimental section.

A promising alternative to extensive sampling is the unscented transform [19]. The unscented transform is a method of selecting samples from an original distribution $\boldsymbol{s} \sim p(\cdot)$ such that the target distribution $\tilde{p}(\cdot)$ obtained through a nonlinear transformation $\tilde{p}(\cdot) = f\left(p(\cdot)\right)$ can be approximated by fitting a distribution to the transformed samples $\tilde{\boldsymbol{s}} = f(\boldsymbol{s})$. The samples $\boldsymbol{s}$ are selected according to sigma points. In the presented case, the distribution $p(R|\boldsymbol{o}, \mathcal{D})$ is only one dimensional and we obtain the sigma points $[\mu_{\boldsymbol{o}}, \mu_{\boldsymbol{o}} + \sigma_{\boldsymbol{o}}, \mu_{\boldsymbol{o}} - \sigma_{\boldsymbol{o}}]$. However, the policy update only depends on the expected value of the reward model, which is independent from additional data points on the current predictive mean. Thus, we can remove $\mu_{\boldsymbol{o}}$ from the set of sigma

points without introducing additional error to the estimate of $\mathbb{E}_{p(R|\boldsymbol{o},\mathcal{D})}[\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)]$.

Even more sample efficient than using sigma points would be to emulate the Upper Confidence Bound AF, which considers an optimistic estimate of the reward. To follow this intuition, we always assign $R = \mu_{\boldsymbol{o}} + \sigma_{\boldsymbol{o}}$ and, thus, require only one evaluation per sample location. We also evaluate the analogous Lower Confidence Bound.

Numerical Stability. The computation of the KL between $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ and $\tilde{\pi}(\boldsymbol{\omega})$ can be done either in closed form in case of Gaussian policies, or numerically in the general case. Computing the sample based approximation of the KL requires importance sampling with respect to the policy that the samples $\boldsymbol{\omega}_i$ were drawn from, i.e.,

$$\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \;||\; \tilde{\pi}(\boldsymbol{\omega})\right) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\pi^*(\boldsymbol{\omega}_i|D^*)}{\pi(\boldsymbol{\omega}_i)} \log \frac{\pi^*(\boldsymbol{\omega}_i|D^*)}{\tilde{\pi}(\boldsymbol{\omega}_i)}. \tag{5}$$

Unfortunately, both the sample based approximation as well as the closed form solution can become numerically unstable for high dimensional parameter spaces (e.g. greater than 20).

For high dimensional policies, we can employ a third alternative, if the policy learning method represents the policy update by assigning weights $\boldsymbol{\gamma}_i$ to the policy samples $\boldsymbol{\omega}_i$. In that case numerical instabilities can be alleviated by computing the KL directly in the one dimensional weight space.

The weights $\boldsymbol{\gamma}_i$ are determined using the policy update function. For a Gaussian policy for example, the updated policy $\tilde{\pi}(\boldsymbol{\omega})$ can then be computed through a weighted Maximum-Likelihood estimate. However, we can avoid first fitting $\tilde{\pi}(\boldsymbol{\omega})$ to $\tilde{\boldsymbol{\gamma}}$ and $\pi^*(\boldsymbol{\omega}|\mathcal{D}^*)$ to $\boldsymbol{\gamma}^*$ by computing the KL divergence of the weight vectors them self.

If we denote the weight vector that would generate $\tilde{\pi}(\boldsymbol{\omega})$ as $\tilde{\boldsymbol{\gamma}}_i = f(\boldsymbol{\omega}_i, p(R|\boldsymbol{o},\mathcal{D}))$ and the weight vector obtained through the updated reward model ${\boldsymbol{\gamma}^*}_i = f(\boldsymbol{\omega}_i, p^*(R|\boldsymbol{o},\mathcal{D}^*))$, then we can compute

$$\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathcal{D}^*) \;||\; \tilde{\pi}(\boldsymbol{\omega})\right) \approx \sum_{i=1}^{N} {\boldsymbol{\gamma}^*}_i \log \frac{{\boldsymbol{\gamma}^*}_i}{\tilde{\boldsymbol{\gamma}}_i}, \tag{6}$$

and avoid numerical instabilities. When computing the KL in weight space, the importance sampling is implicit as $\boldsymbol{\gamma}_i \equiv 1$.

Incorporating Human Error. In Section 2.1 we introduced the additionally hyper parameter β, which models the human's imprecision. We can take advantage of the estimate of the human uncertainty by not including it in our prediction of the reward for EPD. The sample rewards for example in the Sigma point case then become $[\mu_{\boldsymbol{o}} + (\sigma_{\boldsymbol{o}} - \sqrt{\beta}), \mu_{\boldsymbol{o}} - (\sigma_{\boldsymbol{o}} - \sqrt{\beta})]$.

4.2 Information Flow

The general information flow of our method is as follows. We start with an uninformed, i.e., zero mean GP for $p(R|\boldsymbol{o},\mathcal{D})$ and we initialize the PS method with an initial policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$. The PS learner then starts performing one iteration of rollouts. After each iteration of rollouts, rewards for the outcomes of the resulting trajectories $\boldsymbol{o} = \phi(\boldsymbol{\tau})$ are requested from the reward learner $R \sim p(R|\boldsymbol{o},\mathcal{D})$. The reward learner then decides whether to ask for expert ratings for any of the outcomes to update its model. In that case, the agent repeats the corresponding episode to present the outcome to the expert. Finally, the reward learner returns the mean estimate of the reward to the PS learner, which uses the rewards to update its policy and start the next iteration.

5 Background

In this section we provide compact background information on components of the proposed algorithms that are necessary to give a complete picture of the proposed approach.

5.1 Relative Entropy Policy Search

We pair our reward learning algorithm with the recently proposed Relative Entropy Policy Search (REPS) [31]. REPS is a natural choice for a PS method to be combined with an active learning component as it has been shown to work well on real robot problems [9] and is designed to 'stay close to the data'. Thus, previous expert queries will remain informative. A distinctive feature of Relative Entropy Policy Search (REPS), is that its successive policies vary smoothly and do not jump in the parameter space or the context space. This behavior is a beneficial characteristic to increase compliance with our proposed reward learning approach, as we are only able to predict correct rewards within observed regions of the parameter space. To constrain the change in the policy, REPS limits the Kullback-Leibler divergence between a sample distribution $q(\boldsymbol{s},\boldsymbol{\omega})$ and the

next distribution $\pi(\boldsymbol{\omega}|\boldsymbol{s})\mu^\pi(\boldsymbol{s})$

$$\epsilon \geq \sum_{\boldsymbol{s},\boldsymbol{\omega}} \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s}) \log \frac{\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{\omega})}. \tag{7}$$

For the complete optimization problem and its solution we refer to the original work [31].

5.2 Dynamic Movement Primitives

A popular use case for PS methods is to learn parameters of trajectory generators such as Dynamic Movement Primitives (DMPs) [17]. The resulting desired trajectories can then be tracked by a linear feedback controller. DMPs model trajectories using an exponentially decreasing phase function and a nonlinear forcing function. The forcing function excites a spring damper system that depends on the phase and is guaranteed to reach a desired goal position, which is one of the parameters of a DMP. The forcing function is modeled through a set of weighted basis functions $\boldsymbol{\omega}\Psi$. Using the weights $\boldsymbol{\omega}$ of the basis functions Ψ as parameters, we can learn parametrized joint trajectories, and an increasing number of basis functions results in an increased flexibility of the trajectory. We use DMPs for our simulated robot experiments.

5.3 Bayesian Optimization for RL

An alternative approach to learning control parameters $\boldsymbol{\omega}$ using PS methods is to model $p(R|\boldsymbol{s},\boldsymbol{\omega})$ directly and to use Bayesian Optimization (BO) instead of the PS learner. However, the approach proposed in this paper introduces a layer of abstraction which allows us to learn a mapping of only a low-dimensional input space to the reward, as we only have to map from outcomes to reward as opposed to mapping state-action to reward. As BO methods are global methods, they are more susceptible to the curse of dimensionality than PS methods which are local methods. As a result, the mapping $p(R|\boldsymbol{s},\boldsymbol{\omega})$ which the standard BO solution uses is considerably more difficult to learn than the mapping of the modular approach that we are proposing.

The BO approach would also require expert ratings for every sample, while the proposed approach requires only occasional human feedback.

5.4 Acquisition Functions

In the following we present four Acquisition Function (AF) schemes taken from Hoffman et al. [16] that we used to optimize the model of the reward function.

Probability of Improvement The Probability of Improvement (PI) [25] in its original formulation greedily searches for the optimal value of the input parameter that maximizes the function. An adapted version of PI balances the greedy optimization with an exploration-exploitation trade-off parameter ξ. The adapted version is given by

$$\mathrm{PI}(\boldsymbol{o}) = \Phi\left(\frac{\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi}{\sigma(\boldsymbol{o})}\right),$$

where $\boldsymbol{o}^*$ is the best sample in the training set $\mathcal{D}$ and $\Phi(\cdot)$ is the normal cumulative density function. The exploration-exploitation trade-off parameter ξ has to be chosen manually.

Expected Improvement Instead of finding a point that maximizes the PI, the Expected Improvement (EI) [28], tries to find a point that maximizes the magnitude of improvement. Thus, it does not only try to improve local maxima but also considers maxima in different regions and is less greedy in the search of an optimal reward R.

$$\mathrm{EI}(\boldsymbol{o}) = (\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi)\,\Phi(M) + \sigma(\boldsymbol{o})\rho(M),$$

if $\sigma(\boldsymbol{o}) > 0$ and zero otherwise, where $\rho(\cdot)$ is the normal probability density function. M is given by

$$M = \frac{\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi}{\sigma(\boldsymbol{o})}.$$

The EI acquisition function shares the tuning factor ξ for the exploitation-exploration trade-off with the PI, where a suggested value is $\xi = 0.01$ [16].

Upper Confidence Bound The Upper Confidence Bound directly uses the mean and standard deviation of the reward function at the sample location to define the acquisition function. An adapted version of the UCB function [37] is given by

$$\text{GP-UCB}(\boldsymbol{o}) = \mu(\boldsymbol{o}) + \sqrt{v\gamma_n}\sigma(\boldsymbol{o}),$$

where recommended values $v = 1$ and $\gamma_n = 2\log(n^{d/2+2}\pi^2/3\delta)$ with $d = \dim(\boldsymbol{o})$ and $\delta \in (0,1)$ are given by Srinivas et al. [37].

GP Hedge As each of the above acquisition functions lead to a characteristic and distinct sampling behavior, it is often not clear which acquisition function should be used. Portfolio methods, such as the GP-Hedge [16], evaluate several acquisition functions before deciding for a sample location. Given a portfolio with J different acquisition functions, the probability of selecting

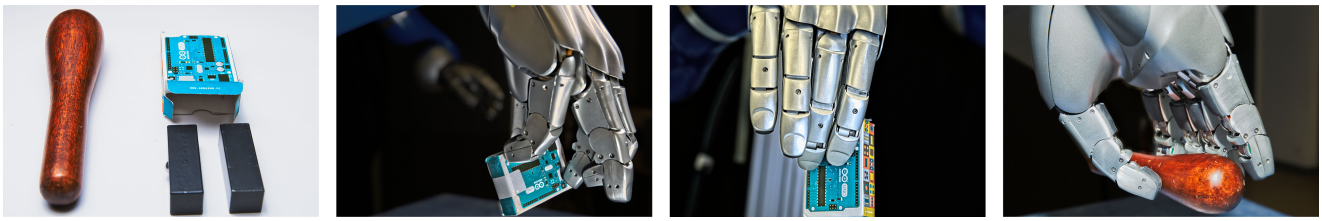

Fig. 3: Examples of different grasps and their categorization. Grasps count as failed if the object is either not picked up at all or if small perturbations would make the object drop. Grasps that are stable but do not keep the original orientation of the object count as OK but not successful grasps. Grasps that are both stable and keep the original orientation count as successful grasps. From Left to right: Pestle and paper box (filled with metal bars); Failed grasp, not robust against perturbations; Mediocre grasp, stable but incorrect orientation; Good grasp, stable with intended orientation.

acquisition function j for the sample $n+1$ is given by the softmax

$$p(j) = \frac{\exp(\eta g_n^j)}{\sum_{i=1}^{J} \exp(\eta g_n^j)},$$

where $\eta > 0$ is the temperature of the soft-max distribution. The gains vector $\boldsymbol{g}$ is initialized to zero, $g_0^{1:J} = 0$, before taking the first sample, and is then updated with the cumulative reward gained by the selected acquisition function, i.e., $g_{n+1}^j = g_n^j + \mu(\boldsymbol{o}_{n+1}^j)$, where $\boldsymbol{o}_{n+1}^j$ is the sample point nominated by acquisition function j. For all other gains the value does not change, i.e., $g_{n+1}^{i \neq j} = g_n^i$.

6 Evaluations

In this section we show evaluations of the proposed active reward learning approach. For all simulation experiments, unless stated otherwise, we tested each setting ten times. We first evaluate a set of traditional AFs against each other and then proceed to compare the best traditional AF to EPD. While on some of the simulated tasks we use a noisy oracle as stand-in for a human expert, all real robot experiments where performed with a human expert. The human experts were the authors of the paper.

6.1 Five Link Reaching Task

To allow extensive evaluation of the proposed methods, we implemented a simulated reaching task. On this task an analytical reward function is readily available such that we can compare a learned reward function against the hard coded one. Additionally, this allows us to use a noisy version of the hard coded reward function as a stand-in for human experimenters and, thus, perform far more repetitions of the experiments. A planar robot consisting of five links connected by rotary joints was controlled in joint space using Dynamic Movement Primitives (DMP) [17], as described in Section 5.2. If not stated otherwise, we used 20 basis functions per joint, resulting in a total of 100 parameters that had to be learned. The hand coded reward function was given by $R(\boldsymbol{p}_r) = 1000 - 100||\boldsymbol{p}_r - \boldsymbol{p}_g||$, where $\boldsymbol{p}_r$ was the position of the robot's end effector and $\boldsymbol{p}_g$ was the desired target position. However, we increased the difficulty of the task for the reward learning component by not supplying the outcome features in task space, but rather in joint space, i.e., the GP had to model the forward kinematics to predict the reward, making the problem both non-convex and high-dimensional (five dimensional mapping).

To allow extensive and consistent evaluation of all parameters of the presented approach, we programmed a noisy expert, which returned the reward with additive white noise (standard deviation was 20). In Fig. 7 we present results that compare the coded noisy expert approach to actually querying a human expert and show that the behavior is comparable. The human expert could give rewards on a vertical bar through a graphical interface.

6.1.1 Evaluation of Acquisition Functions

The evaluation of the available AFs given in Fig. 4 shows that even though there was only limited difference in the asymptotic performance, there was considerable difference in the sample efficiency. Especially the GP-UCB AF asks for many user queries. While PI had the lowest asymptotic performance, as it is the most greedy of the presented AFs, it also required the lowest number of user queries. This behavior makes it an interesting candidate when trying to minimize human interactions.

6.1.2 Evaluation of Uncertainty Threshold

To optimally trade off the number of queries and the agents performance we need to set the uncertainty

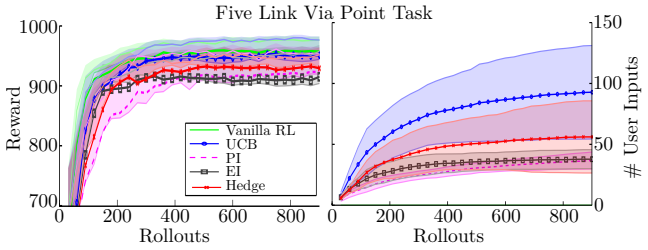

Fig. 4: We evaluated our approach on a programmed, but noisy expert to emulate human expert input. Vanilla RL REPS queries the programmed expert for every sample, while our approach builds a model of the reward function and only queries the expert occasionally.

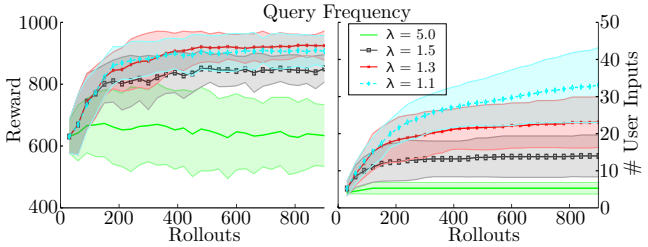

Fig. 5: We evaluated the impact of the threshold factor λ that influences when and how often we sample. Lower values of λ led to better converged performance but required more user interaction.

threshold trade-off parameter $\lambda < \sigma(\boldsymbol{o})/\sqrt{\beta}$. This parameter expresses how certain we require our algorithm to be that a proposed query is not explained by the estimated observation noise. If we choose λ to be greater than 1, we require our estimate to improve on the observation noise. Fig. 5 show the effects of adjusting λ. The performance remained stable up to $\lambda = 1.3$ and started degrading with $\lambda = 2$. Changing the order of magnitude of λ resulted in a failure to learn the task. While the effects of setting $\lambda = 1.5$ on the performance were moderate, the number of queries were reduced by about 50% when compared to $\lambda = 1.3$.

6.1.3 Evaluation of Sparse Sampling

In order to minimize human interaction, we stop improving the GP in each iteration if the outcome that maximizes the AF has already been queried before, instead of selecting the next best outcome according to the AF. The results in Fig. 6 show that sparse sampling leads to equally good asymptotic performance but requires considerably less expert interactions.

6.1.4 Evaluation of Direct Learning

The premise of this paper is that while BO can be used to efficiently learn the mapping of outcome to reward,

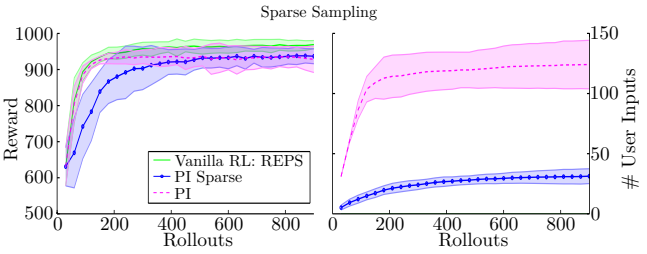

Fig. 6: Results of comparing our acquisition algorithm to the standard acquisition algorithm. Our algorithm collects sparse user queries and does not ask for any user queries after a PS iteration if the outcome that maximizes the AF has already been queried before.

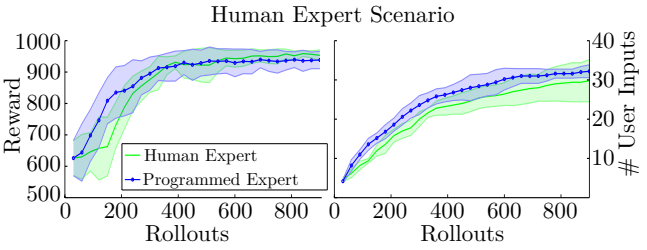

Fig. 7: We validated our approach of using a noisy programmed expert as substitute for a human expert on the simulated tasks. The results show that both experts yielded similar learning behavior.

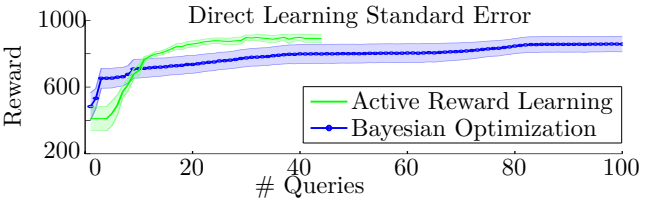

Fig. 8: Comparison of ARL and directly learning the reward from the control parameter $\boldsymbol{\omega}$ using BO. In this figure we plot the standard error as opposed to the standard deviation. ARL performs significantly better after 50 queries ($p < 0.05$).

it would be too sample intensive to directly learn the mapping from parameter space to outcome. We validated this premise by comparing our joint approach to directly learning the reward from the control parameters (i.e., from the basis function weights $\boldsymbol{\omega}$). The results in Fig. 8 show that BO did not converge to a good solution within 50 expert queries. Both methods used PI.

6.1.5 Comparison to Inverse Reinforcement Learning

We compared our algorithm to the Maximum Entropy IRL approach [41] on a via point task in two different scenarios. Trajectories generated by DMPs had to pass one or two via points, respectively (20 dimensional action space for each). For this comparison only, we re-

		#UI = 5	#UI = 10	#UI = 20
ARL	1VP	**995±2.48**	**999±0.25**	**999±0.25**
IRL	1VP	983 ± 3.31	977 ± 6.33	984 ± 3.04
ARL	2VPs	970 ± 13.1	**998±1.13**	**999±0.17**
IRL	2VPs	**994±1.88**	996 ± 0.476	996 ± 0.56

Table 2: Comparison of IRL and ARL. We compare the methods on two via point (VP) tasks with one and two VPs. The columns show the achieved performance (mean and standard deviation) after five, ten or 20 user interactions (UIs).

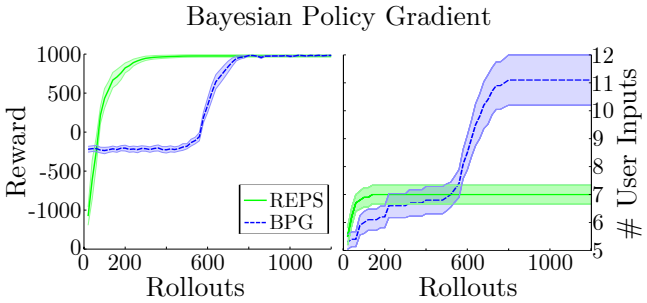

Fig. 9: Performance of REPS and BPG on a via point task.

laxed our assumption that no demonstrations are available and provided the IRL approach with (imperfect) demonstrations that passed close to the via point (at most 0.1 units away). In Table 2, we show the mean and standard deviation of the best policy reached for either five, ten or 20 user interactions (UI). In our approach, UIs are ratings while in IRL UIs are demonstrations. The results show that our approach yields competitive results while not requiring access to demonstrations.

6.1.6 Alternative Policy Learners

While we used REPS for most of our experiments, the proposed framework is not limited to a specific RL method. To investigate the compatibility with other methods, we compared our framework using REPS with our framework in combination with a Bayesian Policy Gradient (BPG) method [13] on a via point task with one via point (20 dimensional action space). BPG models the gradient of the policy using a GP and uses the natural gradient in the policy update. The results in Fig. 9 show that both methods were able to learn the task in combination with our approach.

6.2 Evaluation of Expected Policy Divergence

The previous results demonstrate the compatibility of the presented framework with traditional AFs and establish PI as baseline. In this section, we evaluate and compare the proposed EPD to the baseline PI. We repeat the original experiments on the simulated five link task and provide additional evaluations on a simulated pendulum swing-up task as well as a real robot pendulum swing-up task.

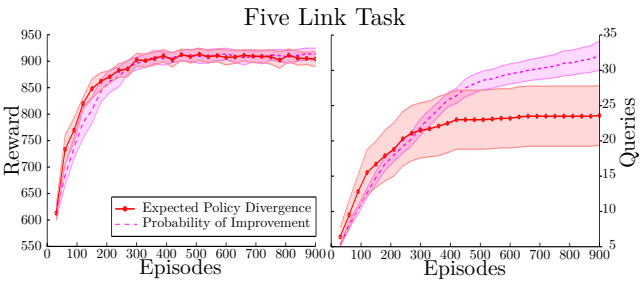

Fig. 10: Performance of EPD ($\kappa = 0.02$) and PI ($\lambda = 1.1$) on the five link task. EPD and PI show similar reward while EPD requires less total user inputs. Since PI does not have a notion of a policy, it keeps optimizing the reward model after the policy has converged.

6.2.1 Five Link Task EPD vs PI

To compare Expected Policy Divergence to PI on the five link task, we repeated the experiment reported above. We chose the tuning parameter $\lambda = 1.1$ for PI and the KL-threshold $\kappa = 0.02$ for EPD such that the algorithms achieve similar performance and we can perform a fair comparison of their querying behavior. Fig. 10 shows that EPD and PI perform equally well in terms of achieved reward but EPD requires less user queries. The user query plot also shows that PI keeps requesting user input after the policy has reached asymptotic performance, while EPD requests more user inputs in early iterations and stops querying after the policy has reached convergence (around episode 400). This behavior is due to the fact that EPD is not trying to optimize the reward model, but instead tries to maximize the information gain for the policy.

6.2.2 Five Link Task Without Constraints

In the previous five link task experiments, we employed the sparse sampling strategy described in Section 3.1, i.e., the AF would stop requesting queries if the currently best candidate had already been sampled before. Additionally, we limited the history of rollouts that can be queried, to the last iteration. Limiting the history is beneficial for all AFs that try to optimize the reward model, since they cannot differentiate between regions of the outcome space that are relevant to the policy and regions of the outcome space where the policy has negligible probability mass. However, improving the reward model in an area of the outcome space without policy probability mass cannot influence the learning process and results in inefficient query strategies. EPD, on the

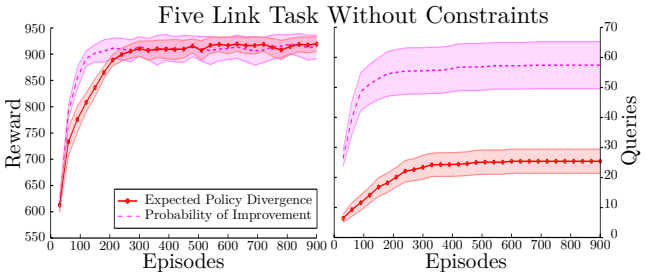

Fig. 11: Performance of EPD ($\kappa = 0.02$) and PI ($\lambda = 1.5$) on the five link tasks without additional constraints. Traditional AFs require additional constraints to be sample efficient in combination with an underlying learning method. EPD's behavior remains almost unaffected from lifting these constraints.

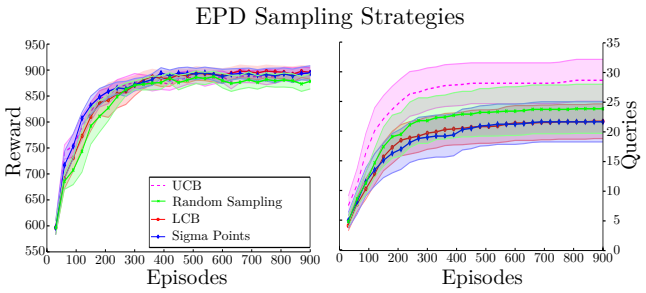

Fig. 12: Performance of different EPD sampling strategies. Sigma point sampling and LCB sampling require the least amount of user queries while delivering competitive performance.

other hand, aims to only improve the reward model where it has influence on the policy learning process. The results in Fig. 11 show that lifting the constraints helps improve the performance for both EPD and PI. Both methods achieve similar asymptotic performance with PI reaching the optimum earlier. However, PI requires almost twice as many samples as before while the querying strategy of EPD remains mostly unaffected.

6.2.3 EPD Sampling Strategies

We evaluated the different sampling strategies discussed in Section 4. Using a sigma point sampling scheme (without the mean) gives the best results in terms of the required user queries while performing well. In this experiment, LCB performs similarly well while UCB requires more user queries to achieve the same result. The results show that random sampling with ten samples also performs well. However, it is computationally more expensive than the other strategies.

6.2.4 Simulated Pendulum Swing-up

We further evaluated the relative performance of the proposed EPD to the performance of PI on a simulated pendulum swing-up task. In this simulated task, a pole of length 0.5m and mass 10kg, distributed equally along the pendulum length, is initially hanging down, mounted to a rotational joint. The joint is actuated and has a friction coefficient of 0.3. The controlled joint can exert a maximum torque of 30Nm, which is insufficient to directly swing up the pendulum to an upright position. Instead, the robot has to first swing the pendulum in the opposing direction to build up enough momentum to swing it up completely. The simulated pendulum swing-up task uses a one dimensional feature, which is the negative sum of the pole tip position to the upright position,

$$\phi_1(\boldsymbol{\tau}) = -\sum_{t=1}^{T} 1 - \cos(\boldsymbol{\theta}_t + \pi),$$

where $\boldsymbol{\theta}$ is the angle of the controlled joint ($\boldsymbol{\theta} = 0$ in the suspended position) and $T = 250$. The programmed reward expert also used this feature as a reward signal, with additional white noise.

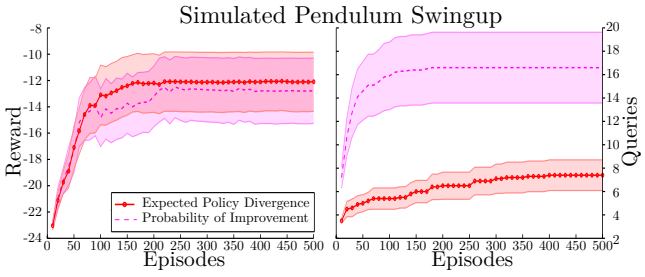

Fig. 14: Performance of EPD ($\kappa = 0.001$) and PI ($\lambda = 1.1$) on the simulated pendulum swing-up task. EPD achieves slightly better performance while requiring less than half the number of user queries.

Fig. 14 shows the comparison of EPD and PI. The results show that EPD has slightly better performance and requires less than half the number of user inputs that PI requires to find the solution. The large variance in the asymptotic performance is due to the fact that some solutions overshoot the goal position initially before achieving the upright position. In the real robot experiment, we add a second feature to the reward computation which helps mitigating this problem. For this experiment the KL threshold for EPD was $\kappa = 0.001$ and the sampling threshold for PI was $\lambda = 1.1$. In each iteration, ten new rollouts were performed. As the pendulum swing-up task is multi-modal, it was solved using HiREPS [9], starting with 100 options. Using HiREPS allowed the agent to learn both swing-ups, clockwise and counter-clockwise.

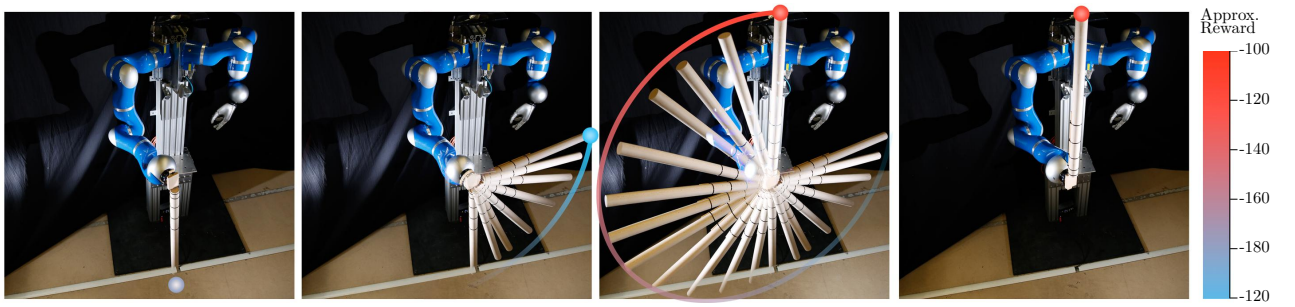

Fig. 13: Picture of the robot performing the swing-up task. The initially suspended pendulum is first counter-clockwise to build momentum and then swung clockwise to the upright position. The heat map gives an indication of the approximate trajectory reward if the swing-up would reach up to the indicated position.

6.2.5 Real Robot Pendulum Swing-up

After comparing EPD and PI on the simulated pendulum swing-up task, we evaluated EPD on a real robot pendulum swing-up task. To learn the reward model we provided the same feature as in the simulated task, with an additional feature representing the sum of the kinetic and potential energy at each time point

$$\phi_2(\boldsymbol{\tau}) = \sum_{t=1}^{T} \frac{1}{2} m v_t^2 + m\left(l - cos(\boldsymbol{\theta}_t + \pi)\right).$$

such that the joined feature vector was $\phi(\boldsymbol{\tau}) = [\phi_1(\boldsymbol{\tau}), \phi_2(\boldsymbol{\tau})]^T$. Adding the energy as feature helps to differentiate trajectories that exhibit the desired behavior of performing a pre-swing from those that try to directly swing up the pendulum in the early stages of learning. While it is difficult to provide a hard-coded target value for this feature, it is easy to learn the desired value from human ratings. The reported reward for the real robot experiment is computed only from the ϕ_1, analogously to the simulated swing-up task. In the real robot experiments $T = 10e4$s, the length of the pole was $l = 1$m, with a mass of $m = 10kg$.

Fig. 15 shows the results of three trials on the real robot task. The results show that the robot learned to swing up the pendulum in all three trials. In the last trial, the robot learned a double pre-swing before swinging up the pendulum. For the real robot experiment the KL threshold for EPD was $\kappa = 0.005$. As in simulation, ten new rollouts were performed per iteration.

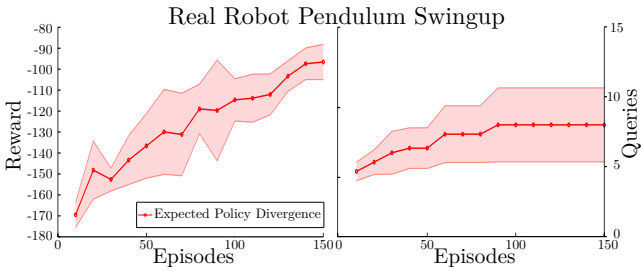

Fig. 15: Performance of Expected Policy Divergence on the real robot pendulum swing-up task. EPD learns to successfully swing up the pendulum on the real robot with an average of about eight user queries. Results averaged over three trials.

6.3 Robot Grasping

We used the results from Section 6.1 to set the parameters for the real robot experiment (we use PI and set $\lambda = 3$). We learned the policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with 15 samples per iteration for a total of 10 iterations and we repeated the experiment three times. The control parameters of the policy were the 15 joints of the five finger DLR hand, which is mounted to a 7 DOFs KUKA lightweight arm as shown in Fig. 1. We considered the task of blind grasping as described by Dang and Allen [8], where no object information was available and we did not have visual feedback, i.e., we did not have information about the contact points. Instead, we calculated the forces in the finger tips through the joint torques and and the hand kinematics. We used the finger tip force magnitudes as outcome features which were used by the GP to model the reward function.

Evaluating the real robot experiments presented the problem of a success metric. As we did not have a 'correct' reward function that we could evaluate the learned reward function against, we resorted to introducing three label categories which were only used for the evaluation after finishing the experiments. The scheme presented in Fig. 3 labels grasps as failures with a reward of -1, if the object was not lifted at all or slipped when slightly perturbed. Grasps that were stable but did not keep the intended orientation were given a reward of 0. Finally, grasps that lifted the object and kept the orientation of the object were assigned a reward of 1. These labels and reward values were not used during the learning of either the policy or the reward model

but only used to visualize the robot's learning progress. During the learning phase, the human expert assigned grasp ratings in the range of ±1000.

6.3.1 Learn to Grasp Unknown Object

The object to be grasped was a cardboard box filled with metal weights such that the robot cannot grasp the object with very unstable grasps or by deforming the paper box. The box, shown in Fig. 3, was of size 7.5cm x 5.5cm x 2cm and filled with two metal bars with a combined weight of 350g. The results of three trials presented in Fig. 16 show that the robot learned to perform a successful grasp of the object in all three trials, while only requesting six queries in the first and twelve queries in the second trial. In the last trial, the robot's performance first increased quickly but dropped after 80 episodes (or rollouts), coinciding with a sudden increase of user queries, such that the final number of queries in the last trials was 27. The reason for this unusual behavior was a malfunction of the distal joint of the thumb, which rendered the grasping scheme dysfunctional. As the learner could not reproduce outcomes that led to good rewards, it resorted to finding a different grasp strategy. At the same time, since the GP was presented with new outcome samples in previously unobserved regions of the outcome space, it requested new user queries to model the reward function in the new region of interest.

We compared our learned reward function to a (naive) hand coded reward function based on the same features. The hand coded reward function aimed to reach a total force magnitude over all fingers. Using the programmed reward, the robot was able to reliably pick up the object after the first two trials and in ten out of 15 grasps at the end of the third trial (We expect the robot would have also learned to pick it up reliably in the last trial with more iterations). However, the robot did not pick up the object in a way that kept the original orientation of the object. Encoding such behavior through only force features by hand is challenging. The performance curve of the hand coded reward function shows a slight dip, which is possible as we are not plotting the internal reward but the reward assigned according to the grading scheme introduced in Fig. 3.

6.3.2 Transfer Learned Reward Function to New Object

As we based our reward model only on the finger tip forces, it is modular and can be used for other similar objects. To test these generalization capabilities, we started a new trial with a different object (a pestle as shown in Fig. 3) and initialized the GP with the training data from the first trial of the previous task. For this experiment we only used ratings from the previous trial, and did not ask for additional expert ratings. The pestle that we used for this task was similar in dimensions but different in shape, such that the agent had to learn different joint configurations to achieve similar finger forces that optimized the reward function. The pestle had a length of 18cm, with a 1.5cm radius at the thin end and a 2.25cm radius at the thick end. The results in Fig. 17 show that the agent was able to learn to reliably perform a robust grasp on the pestle, reusing the reward function learned for the box.

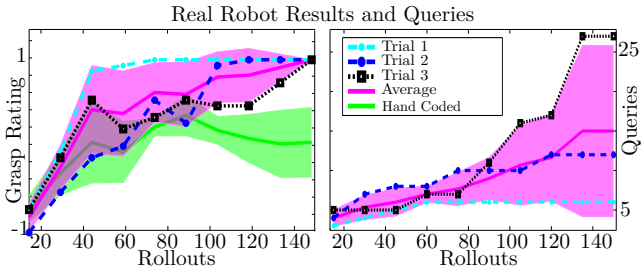

Fig. 16: The real robot successfully learned good grasps in all of the three trials. The grasp rating scheme is described in Fig. 3. In the last trial, one joint failed, represented by a spike in expert queries. The robot successfully adapted the reward function to recover from the hardware failure. We also show the average performance of a naive hand coded reward function.

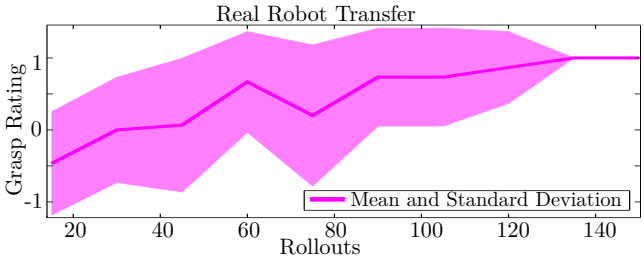

Fig. 17: Performance of one trial on the real robot system. The robot uses the reward function learned in trial one of the original task to learn to grasp a new, unknown object (a wooden pestle as shown in Fig. 3). For this trial, we did not allow any expert queries. The robot successfully learned to grasp the object.

7 Related Work

The term reward shaping describes efforts to adapt reward to increase learning speed while not affecting the policy [30, 10, 4, 36] or to accelerate learning of a new task [22]. Aside from reward shaping, Dorigo and Colombetti [11] have used the term robot shap-

ing to describe efforts of teaching different tasks to a robotic agent. Derived from this terminology the TAMER framework [20] uses the term interactive shaping. There, the agent receives reinforcements from a human, guiding the learning process of the agent without active component. The Advise framework [14] uses the term shaping to describe the process of modeling an oracle and actively requesting binary labels for an agent's action (good or bad). This approach is tailored for discrete settings and the feedback frequency is pre-defined.

In preference learning algorithms, the expert is usually requested to rank the current best against a new execution and the reward function is inferred from these rankings [2, 6, 40]. The method of Akrour et al. [1] can also deal with noisy rankings. Chu and Ghahramani [7] introduced the use of GPs for preference rankings. However, the expert is limited to transmit one bit of information and cannot express strong preferences. Our contribution is to show how a rating based approach with explicit noise model can be used in real robot continuous state-action space problems taking advantage of the stronger guidance through strong preferences (large differences in assigned rewards).

Schoenauer et al. [35] show how a preference based learning algorithm can be used in the cart-pole balancing task with continuous states and discrete actions. However, they require to first build a model of the system dynamics through extensive sampling. In our approach the core idea is that the control of the system and the reward function should be learned simultaneously, which minimizes the total number of rollouts required on the system. If, on the other hand, the number of rollouts on the system to build a model is of no importance, different approaches to learning the reward function are better suited.

The problem of expert noise has also been addressed in Bayesian RL and IRL. In Bayesian RL, Engel et al. [12] have proposed to model the value function through a GP and Ghavamzadeh and Engel [13] have proposed to model the policy gradient through a GP. Especially the policy gradient method is also well suited to work in combination with our approached framework.

Kroemer et al. [24] have used Gaussian Process Regression with a UCB policy to directly predict where to best grasp an object. If demonstrations are available, IRL is a viable alternative. Ziebart et al. [41] have relaxed the assumptions on the optimality of demonstrations such that a reward function can be extracted from noisy demonstrations.

In a combination of IRL and preference learning, Jain et al. [18] have proposed the iterative improvement of trajectories. In their approach, the expert can choose to rank trajectories or to demonstrate a new trajectory that does not have to be optimal but only to improve on the current trajectory. This approach cannot directly be used on learning tasks such as grasping, as a forward model is required. Alternatively, Lopes et al. [26] propose a framework where IRL is combined with active learning such that the agent can decide when and where to ask for demonstrations.

8 Conclusion & Future Work

We presented a general framework for actively learning the reward function from human experts while simultaneously learning the agent's policy. We propose a novel acquisition function which evaluates the change of the policy induced through altering the reward model. Our experiments showed that the learned reward function can outperform hand-coded reward functions, generalizes to similar tasks and that the approach is robust to sudden changes in the environment, for example a mechanical failure. The results also show that the proposed acquisition function, EPD, outperforms traditional AFs and works well on simulated and real robot experiments. In the future we plan to investigate how different kernel functions affect the learning process and study how well non-expert users can 'program' reward functions with the presented framework.

Acknowledgments

The authors want to thank for the support of the European Union projects #FP7-ICT-270327 (Complacs) and #FP7-ICT-2013-10 (3rd Hand).

References

1. Riad Akrour, Marc Schoenauer, and Michele Sebag. Preference-based policy learning. In *Machine Learning and Knowledge Discovery in Databases*. 2011.
2. Riad Akrour, Marc Schoenauer, and Michele Sebag. Interactive robot education. In *European Conference on Machine Learning Workshop*, 2013.
3. Ravi Balasubramanian, Ling Xu, Peter D Brook, Joshua R Smith, and Yoky Matsuoka. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *IEEE Transactions on Robotics*, 2012.
4. Jeshua Bratman, Satinder Singh, Jonathan Sorg, and Richard Lewis. Strong mitigation: Nesting search for good policies within search for good reward. In *International Conference on Autonomous Agents and Multiagent Systems*, 2012.

5. Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *International conference on Human-Robot Interaction*, 2012.
6. Weiwei Cheng, Johannes Fürnkranz, Eyke Hüllermeier, and Sang-Hyeun Park. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*. 2011.
7. Wei Chu and Zoubin Ghahramani. Preference learning with Gaussian processes. In *International Conference on Machine Learning*, 2005.
8. Hao Dang and Peter K Allen. Learning grasp stability. In *International Conference on Robotics and Automation*, 2012.
9. Christian Daniel, Gerhard Neumann, and Jan Peters. Learning Sequential Motor Tasks. In *International Conference on Robotics and Automation*, 2013.
10. Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *International Conference on Autonomous Agents and Multiagent Systems*, 2012.
11. Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 1994.
12. Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, 2005.
13. Mohammad Ghavamzadeh and Yaakov Engel. Bayesian policy gradient algorithms. *Advances in Neural Information Processing Systems*, 2007.
14. Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, 2013.
15. Harold W Hake and WR Garner. The effect of presenting various numbers of discrete steps on scale reading accuracy. *Journal of Experimental Psychology*, 1951.
16. Matthew D Hoffman, Eric Brochu, and Nando de Freitas. Portfolio allocation for Bayesian optimization. In *Conference on Uncertainty in Artificial Intelligence*, 2011.
17. Auke Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems*. 2003.
18. Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems*, 2013.
19. Simon J Julier and Jeffrey K Uhlmann. A new extension of the kalman filter to nonlinear systems. In *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.
20. W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *International Conference on Knowledge Capture*, 2009.
21. Jens Kober, Betty J. Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *Intelligent Robots and Systems*, 2008.
22. George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *International Conference on Machine Learning*, 2006.
23. Petar Kormushev, Sylvain Calinon, and Darwin Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *Intelligent Robots and Systems*, 2010.
24. Oliver Kroemer, Renaud Detry, Justus Piater, and Jan Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 2010.
25. Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 1964.
26. Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*. 2009.
27. George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 1956.
28. Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 1978.
29. Andrew Ng and Adam Coates. Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, 1998.
30. Andrew Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Internation Conference on Machine Learning*, 1999.
31. Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *National Conference on Artificial Intelligence*, 2010.
32. Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. 2006.

33. N. Ratliff, A. Bagnell, and M. Zinkevich. Maximum Margin Planning. In *International Conference on Machine Learning*, 2006.
34. Nathan Ratliff, David Silver, and Andrew Bagnell. Learning to Search: Functional Gradient Techniques for Imitation Learning. *Autonomous Robots*, 2009.
35. Marc Schoenauer, Riad Akrour, Michele Sebag, and Jean-christophe Souplet. Programming by feedback. In *International Conference on Machine Learning*, 2014.
36. Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *Autonomous Mental Development*, 2010.
37. Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *Internation Conference on Machine Learning*, 2010.
38. Raúl Suárez Feijóo, Jordi Cornellà Medrano, and Máximo Roa Garzón. Grasp quality measures. 2012.
39. Andrea L Thomaz and Cynthia Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 2008.
40. Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems*, 2012.
41. Brian Ziebart, Andrew Maas, Andrew Bagnell, and Anind Dey. Maximum entropy inverse reinforcement learning. In *Conference on Artificial Intelligence*, 2008.