Arndt, Karol; Struckmeier, Oliver; Kyrki, Ville

Domain Curiosity: Learning Efficient Data Collection Strategies for Domain Adaptation

Domain Curiosity: Learning Efficient Data Collection Strategies for Domain Adaptation

Karol Arndt[1], Oliver Struckmeier[1], and Ville Kyrki[1]

Abstract— Domain adaptation is a common problem in robotics, with applications such as transferring policies from simulation to real world and lifelong learning. Performing such adaptation, however, requires informative data about the environment to be available during the adaptation. In this paper, we present *domain curiosity*—a method of training exploratory policies that are explicitly optimized to provide data that allows a model to learn about the unknown aspects of the environment. In contrast to most curiosity methods, our approach explicitly rewards learning, which makes it robust to environment noise without sacrificing its ability to learn. We evaluate the proposed method by comparing how much a model can learn about environment dynamics given data collected by the proposed approach, compared to standard curious and random policies. The evaluation is performed using a toy environment, two simulated robot setups, and on a real-world haptic exploration task. The results show that the proposed method allows data-efficient and accurate estimation of dynamics.

I. INTRODUCTION

Since the advent of reinforcement learning methods capable of handling complex, continuous state spaces [1]–[4], the ability to train control policies using only data coming from interacting with the environment has gained interest in the robotics community. Modern reinforcement learning methods, however, still require extensive interactions with the environment before learning useful behaviours, which may take from hours [2] to months [5] of real-time interaction. To reduce the dependence on large amounts of real-world data, a policy is often trained in a simulated environment, where the general structure of the problem is learned, and then deployed in the physical world. This scenario is known as *domain transfer*.

Domain transfer in reinforcement learning has been extensively studied, especially in the context of sim-to-real transfer in robotics [5]–[7]. Multiple approaches to this problem have been proposed in the past, such as training a robust policy that performs well across a range of dynamics conditions [8], [9], training a policy conditioned on the belief over dynamics [10], or training a policy that is able to adapt to new dynamic conditions [11], [12]. However, robust policies do not generalize well when the possible range of dynamics is wide [13], especially in situations where the task requires high precision or the optimal actions depend heavily on the dynamics [6].
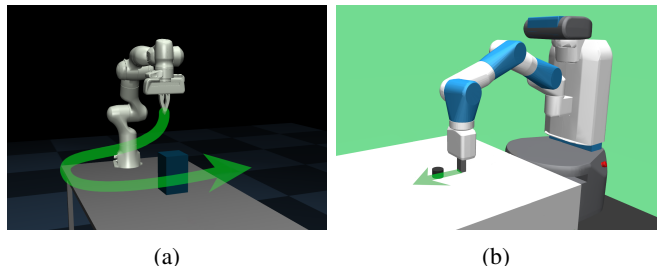
Fig. 1: The proposed method learns efficient strategies to collect data for dynamics estimation, as shown in a haptic exploration setup (a) and on a sliding setup with unknown friction parameters (b).

Adaptation and domain identification approaches, on the other hand, use data from the target environment; this dataset has to be complex enough to allow the model to identify the dynamics parameters, and yet small enough to be feasible to collect on physical hardware within a reasonable time. The data collection process also needs to be safe to perform on physical hardware which rules out random policies. In addition, dynamics identification requires the actions to be identifiable which poses a problem e.g. with kinesthetic human demonstrations, where only states may be directly measured. Gathering the data for dynamics adaptation is thus a central problem for domain adaptation and identification.

In this paper, we propose the *domain curiosity* approach to learning a policy that efficiently collects such adaptation data. In contrast to previously proposed intrinsic rewards in the domain adaptation context, our method rewards the agent for providing data that allows a dynamics model to improve its predictions about the environment, instead of rewarding prediction error. This is achieved by jointly training a meta-learned adaptable dynamics model together with an exploration policy.

We experimentally evaluate the method on a toy environment and on two simulated robotics setups: a sliding task under unknown friction and a haptic exploration task, where the robot's goal is to locate the position of an object in the scene. We also deploy the policy for the latter setup on a real-world Franka Panda robot arm.

The contributions of this paper are (1) proposing a novel framework for training a curious policy that maximizes the agent's information gain about the domain-specific aspects the environment, (2) providing experimental evaluation in simulated and real world environments, (3) showing that the proposed domain curious approach is able to reliably provide data which helps to identify the dynamic conditions in the environment and to perform better next state prediction.

II. RELATED WORK

A. Curiosity in reinforcement learning

The idea of intrinsic, curiosity-based exploration was first described by Schmidhuber [14]. In order to focus on epistemic (*i.e.,* learnable) uncertainty, this method was later modified to reward model improvement [15], [16], model parameter updates [17], work in feature space [18] and reward disagreement between ensemble dynamics models [19], [20], instead of operating on raw state predictions.

In robotics, curiosity- and uncertainty-based rewards were successfully used with Bayesian optimization to improve exploration in sparse reward settings [21] and in exploration tasks [22]; however, these methods did not include any domain variability or domain adaptation.

Our method learns to focus on epistemic uncertainty by rewarding model improvement, similarly to [15]; however, our method is designed for a domain adaptation scenario, which allows it to distinguish prediction errors induced by lack of knowledge about the domain from pure noise. In contrast to [19], our approach performs inner-loop model updates (*model adaptation*) and rewards performance improvements caused by these updates, instead of attempting to learn a global model for all domains. This is distinction is crucial, since, to a single dynamics model, domain uncertainty would present itself as aleatoric rather than epistemic uncertainty, as the non-adaptable dynamics model cannot further reduce the prediction error for a particular domain beyond learning the expected outcome over all domains, or modelling its distribution (which includes aleatoric uncertainty).

Our method shares the basic theoretical formulation with VIME [17]. VIME maximizes the information gain of an agent about a distribution of dynamics models in addition to the extrinsic reward from the environment. The information gain is measured using variational inference with a reward term based on the KL divergence between two following hidden states of a model. This method, however, considers only a single training domain, while our approach combines it with the meta-learning formulation and applies it to the multi-domain reinforcement learning scenario.

B. Exploration in meta-reinforcement learning

In meta-RL, various ways of learning exploration strategies have been previously explored. The original model-agnostic meta-learning (MAML; [23]) formulation only optimizes for post-adaptation performance, without explicitly optimizing the meta-policy to provide useful samples for adaptation. Despite this shortcoming, MAML has been successfully applied to domain adaptation and sim-to-real transfer [11], [12]. This problem was later addressed in [24] and [25], where the agent was additionally incentivised to provide useful trajectories during adaptation, and in [26], where an uncertainty-aware Bayesian model was used as the policy in combination with the maximum entropy objective function [2], [27]. This resulted in more stable and more consistent training procedure.

More recently, VariBAD [10] was shown to perform close to Bayes-optimal exploration in simple gridworlds by performing approximate variational inference over a hidden parameter and conditioning a dynamics model on that hidden parameter. The method implicitly learns to explore by using this hidden parameter to meta-learn a policy on an unknown task based on previous experience. However, it does not directly include the prediction accuracy in the reward function and requires an extrinsic task reward. In contrast, our method directly encourages collecting useful data by rewarding the correction to the model's predictions.

Zhang *et al.* [28] described a meta-learning framework based on similar grounding to VIME; however, in the practical implementation, this approach was simplified to embedding a basic surprise-based curiosity reward inside a meta-reinforcement learning agent.

A limitation of previous meta-RL methods is the dependence on on-policy data. Rakelly *et al.* [29] address this with a stochastic component performing inference over a latent context variable to encourage exploration and ultimately faster adaption to new tasks. Domain curiosity on the other hand jointly learns the task and a dynamics model in a way that promotes exploration of informative components of the environment to achieve the same effect.

III. METHOD

A. Definitions and problem formulation

The domain adaptation problem problem can be formulated as a partially observable Markov decision process (POMDP) of the form $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p_{tr}, p_{s_0}, r, \xi)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, ξ is a vector of dynamics parameters, $p(s_0)$ is the initial state probability distribution, $p_{tr} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Xi \to \mathcal{R}$ represents the transition probabilities conditioned on ξ, and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function. By following a policy $\pi(a|s)$, the agent collects a set of trajectories: $\mathcal{D} = \{\tau_0, ..., \tau_N\}$, where each trajectory is a sequence of states and actions: $\tau = (s_0, a_0, s_1, a_1, ..., a_{T-1}, s_T)$.

Then, the specific problem addressed in this paper can be formulated as follows: find a policy $\pi(a|s)$ that collects a trajectory dataset $\mathcal{D}$ that maximizes the agent's information gain about the dynamics parameters ξ. More specifically, knowing ξ reduces the entropy of dynamics transitions, $\mathcal{H}(p_{tr}(s, a, s'|\xi)) \leq \mathcal{H}(p_{tr}(s, a, s'))$. Since each individual state transition depends on ξ, so does $\mathcal{D}$: $p(\mathcal{D}|\xi)$, and thus $\mathcal{D}$ can, in turn, be used to estimate ξ: $p(\xi|\mathcal{D})$. All in all, the agent's information gain can be expressed by the entropy difference $\mathcal{H}(p_{tr}(s, a, s')) - \mathcal{H}(p_{tr}(s, a, s'|\mathcal{D}))$.

B. Policy training

The training procedure for Domain curiosity is outlined in detail in Algorithm 1, and an overview of the reward computation procedure is shown in Figure 2. The main idea is to have an adaptable dynamics model of the environment $f_\phi(s, a)$, whose prediction updates can be used as a basis for reward computation. These rewards are then used to train an exploratory policy. Our method assumes that the dynamics model is not available in advance, and trains one from scratch using data collected by the policy.
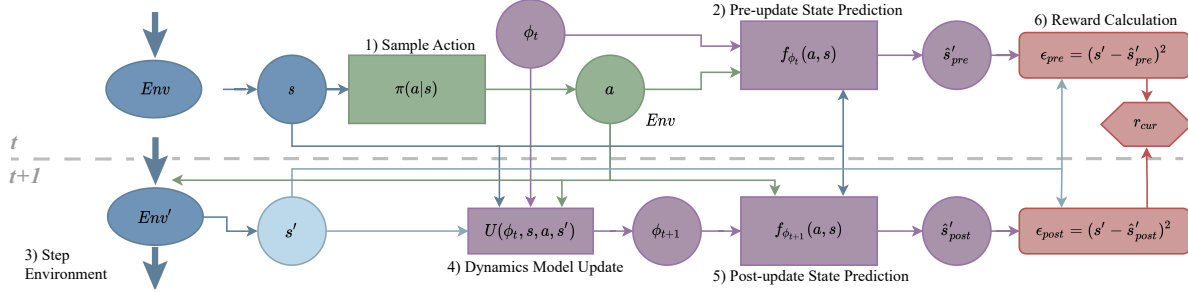
Fig. 2: Overview of the proposed reward calculation method. The elements of the agent are marked in green, the environment—in blue, and the dynamics model in purple.

The algorithm is initialized by randomly setting the policy weights θ and the model weights ϕ, and initializing two empty buffers for storing state transitions: one for the policy $\mathcal{D}_\pi$ and one for the dynamics model $\mathcal{D}_f$ (steps 1–2). On-policy methods like PPO [4] require $\mathcal{D}_\pi$ to be cleared after every policy update; since model training can be performed off-policy, using a single, shared buffer in this context would be wasteful, and could make model training unstable due to data distribution shift.

Each training iteration starts by sampling a random task ξ, which—in the domain adaptation context—defines the dynamics of the system. At the same time we also reset the environment (step 5) and set the belief over the hidden state back to the prior (step 6).

Algorithm 1: Domain curiosity

Result: Parameters θ of a policy that identifies ξ

1   randomly initialize θ and ϕ_0 ;
2   initialize empty replay buffers $\mathcal{D}_\pi$ and $\mathcal{D}_f$;
3   **while** *not done* **do**
4   Sample a random task $\xi_i \sim p(\xi)$;
5   Reset the env; $s \sim p(s_0)$;
6   Reset the model parameters; $\phi_t = \phi_0$;
7   **while** *episode not over* **do**
8   Sample $a \sim \pi_\theta(a|s)$;
9   Calculate pre-update state prediction (Eq. 1) ;
10   Execute a, observe s';
11   Update the dynamics model (Eq. 2) ;
12   Calculate post-update state prediction (Eq. 3) ;
13   Calculate the reward (Eqs. 4 and 5) ;
14   Store (s, a, r, s') in $\mathcal{D}_\pi$ and $\mathcal{D}_f$;
15   **end**
16   Update π_θ with $\mathcal{D}_\pi$ to maximize expected cumulative return;
17   Update ϕ_0 with $\mathcal{D}_f$ to minimize prediction errors over each sequence (Eq. 6);
18 **end**

For each episode, we follow the policy π_θ. We first sample an action from the policy (step 8 in Alg. 1 and step 1 in Fig. 2). We then use the current dynamics model to predict the next state (step 9 in Alg. 1 and step 2 in Fig. 2) as

$$\hat{s}'_{pre} = f_{\phi_t}(s, a). \tag{1}$$

Next, we step the environment using a and observe the true next state s' (step 10 in Algorithm 1 and step 3 in Figure 2). Then, we update the belief over dynamics using the observed transition (step 11 in Alg. 1 and step 4 in Fig. 2) by

$$\phi_{t+1} = U(\phi_t, s, a, s'), \tag{2}$$

where $U(\cdot)$ represents the inner-loop update rule used by the dynamics model (e.g., in case of MAML, U would correspond to performing a step of gradient descent).

To quantify the *improvement* of the dynamics model after observing the transition, we predict the post update state $\hat{s}'_{post}$ using the dynamics model with the *updated* state belief from Eq. 2 (step 12 in Alg. 1 and step 5 in Fig. 2):

$$\hat{s}'_{post} = f_{\phi_{t+1}}(s, a). \tag{3}$$

This effectively evaluates if observing the transition allows the model to make more accurate prediction about it.

While it may seem that having observed the transition would always allow the model to make a good prediction by simply memorizing the value that has just been passed to it, this would only be the case with a model that memorizes the observed samples instead of generally modelling the environment. This was an important consideration for selecting the architecture of the dynamics model; we provide more details behind it in Section III-C. To provide a tangible example, observing a sample from the unit Gaussian distribution does not allow one to make more accurate predictions about future samples from that distribution (assuming the distribution that the sample came from is known). This step allows our method to tell apart prediction errors coming from noise (*aleatoric uncertainty*) and prediction errors coming from insufficient knowledge of the domain the model has been deployed in (*epistemic uncertainty*). It also differentiates the proposed method from standard curiosity-based intrinsic rewards, where the policy is rewarded for model prediction errors, as opposed to rewarding the reduction in the error.

We finally calculate the prediction error for both methods $\varepsilon_{pre} = (s' - \hat{s}'_{pre})^2$ and $\varepsilon_{post} = (s' - \hat{s}'_{post})^2$. The domain curiosity reward is then calculated as the difference between the two prediction errors, corresponding to the reduction in prediction error (step 13 in Alg. 1 and step 6 in Fig. 2)

$$r_{cur} = \varepsilon_{pre} - \varepsilon_{post} \tag{4}$$

To regularize the policy and to make it physically feasible, we also add a control cost penalty

$$r = r_{control} + \lambda r_{cur} \tag{5}$$

The control cost, expressed by the reward term $r_{control}$, penalizes large control forces and—in case of robot environments—large contact forces between the robot and the objects in the environment. The λ hyperparameter is used to balance between control cost and the curiosity reward.

We then store the observed transition (s, a, s') in $\mathcal{D}_f$ in order to update the dynamics model, and (s, a, r, s') in $\mathcal{D}_\pi$ for training the policy. After finishing each episode, we update the policy π_θ and the dynamics model f_ϕ. The policy update can be performed with an arbitrary reinforcement learning algorithm.

C. Dynamics model

Generally speaking, the proposed policy training approach can be used with an arbitrary adaptable meta-learned dynamics model. We found MAML [23] to be impractical due to the large computational cost of performing an update every timestep over a long episode, and due to the potential of memorizing the last-seen sample, which is facilitated through the gradient-descent inner-loop learning rule. We also found the classic LSTM approach [30] to be unsuitable, as the post-update state prediction step in the proposed method requires the calculation of output given the updated hidden state ($f(x_t, h_t)$), whereas, during training, the model is never trained to make such predictions (only $f(x_t, h_{t-1})$); such samples also cannot be directly included in the training procedures, as that creates a direct path in the model between the true value y_t and the model's prediction $\hat{y}$, which encourages the model to "cheat" by simply memorizing the previous observed sample in h_t. This would effectively cause the model to make accurate post-update predictions of observation and/or system noise.

Given these considerations, we selected the backprop Kalman filter (BKF)-based approach proposed in [31]. The model is separated into three parts: dynamics measurement, belief update, and next state prediction. Unlike the recurrent method in [30], the architecture is constructed such that the recurrent hidden state is optimized to only encode information about the dynamics and does not need to store information about previously observed samples. This allows the reward calculation procedure to query arbitrary dynamics beliefs with arbitrary state-action pairs. In this formulation, model parameter vector can be split in two parts: the *static* parameters, which are only updated in the outer loop (all the neural network weights) and the task specific parameters, updated by $U(\cdot)$ (the hidden state of the filter, as defined by the mean and variance of the state belief).

To further facilitate learning over long sequences, we change the loss function to calculate be the sum of the prediction loss over the whole sequence:

$$\mathcal{L} = \sum_{t=0}^{T} \sum_{n=0}^{N} (s'_n - f_{\phi_t}(s_n, a_n))^2 \qquad (6)$$

With this change, it is important to keep the data used to compute the values of ϕ_t separate from the state transitions used to calculate the loss (in contrast to [30], where the same data is used for both tasks, but the target labels are shown

to the model only after it has already made the prediction for their corresponding input). This prevents the model from memorizing the observed samples in the hidden state. While we find it unlikely to occur in practice, it would theoretically still be possible if the selected dimensionality of the hidden states ϕ were too large. We also found it beneficial to add L2 norm regularization on ϕ, in the form of adding $\sum_{t=0}^{T} \phi_t^2$ to the loss term.

IV. EXPERIMENTS

In this section, we present the results of the experimental evaluation in the toy environment, in two simulated robotics setups and on a real-world setup with the Franka Panda robot.

In all experiments, the policy is trained using Proximal Policy Optimization (PPO; [4]), but—in general—the method can be used with an arbitrary reinforcement learning algorithm. During the first updates, the model's predictions and inner-loop updates are random, which can have an adverse effect on learning. To reduce this effect and, effectively, stabilize the training, we only update the dynamics model in the early phases of training, and later gradually start training the policy by linearly increasing the PPO clip coefficient from 0 to the final value. We perform 4 epochs of model updates per each policy update and use a model replay buffer of 25000 episodes in all experiments. The λ hyperparameter, which balances the curious reward and the control cost, is set to 10^4 in all environments.

A. Toy example

To demonstrate the method on a simple example and provide an evaluation with a Bayes-optimal task variable prediction model, we developed a simple environment, *SpotsAndHoneypots*, shown in Figure 3. In this environment, an agent has to learn to explore task specific variables by visiting three *spots* (marked as green, purple and orange circles in Figure 3) that provide noisy estimates of those variables. The environment also includes a *honeypot* that acts as a source of random noise. The state space is $(x, y, \alpha) \in R^3$ where (x, y) is the position of the agent and α is a variable that depends on the current dynamics, ξ. The action space consists of the agent's desired velocity in each direction $a = (v_x, v_y)$. At the beginning of an episode the agent, represented by a black dot, starts in the center of the environment.

Whenever one of the spots is visited by the agent, the third element in the state vector is a noisy measurement of one of the task variables, proportionally to the distance from the center of the spot: $\alpha \sim \mathcal{N}(\mu_\alpha, 10^{-4})$, $\mu_\alpha = (1 - \frac{d}{\rho})\xi_i$, where i is the index of the visited spot, ρ is its radius, and d is the distance to center. Thus, in order to learn about the task-specific aspects of the environment, the agent needs to explore all three spots.

In honeypot, the yellow spot in Figure 3, α is a random value sampled from a zero-mean Gaussian distribution: $\alpha \sim \mathcal{N}(0, 25)$. Similarly to the *spots*, this value is also scaled by the distance to center $(1 - \frac{d}{r})$. The task variable describing the system dynamics, ξ, is a three-element vector, with each element sampled independently from $\mathcal{U}(-8, 8)$.

Method	SpotsAndHoneypots	FetchSlide	PandaFindBox
Random	6.3	0.092	0.21
Task policy	–	0.053	–
Random reach	–	–	0.20
Curiosity	7.7	0.044	0.21
Uniform	2.6	–	–
Domain curiosity	**0.7**	0.042	**0.03**

TABLE I: Task variable prediction error after 50 (SpotsAndHoneypots, FetchSlide) or 200 samples (PandaFindBox).

Method	SpotsAndHoneypots	FetchSlide	PandaFindBox
Random	0.81	$5.18 \cdot 10^{-4}$	$7.8 \cdot 10^{-4}$
Task policy	–	$4.15 \cdot 10^{-4}$	–
Random reach	–	–	$7.8 \cdot 10^{-4}$
Curiosity	0.81	$4.09 \cdot 10^{-4}$	$7.8 \cdot 10^{-4}$
Uniform	0.77	–	–
Domain curiosity	0.68	$4.08 \cdot 10^{-4}$	$7.6 \cdot 10^{-4}$

TABLE II: Mean next state prediction error after observing 50 samples using data collected by each method.

In this environment, a standard 'greedy' curious algorithm is attracted to the yellow honeypot (as shown in Figure 3c), while the proposed method learns a policy that moves around the whole environment, visiting all three spots and thus revealing information about all elements of the task variable ξ, as shown in Figure 3d. This result also experimentally verifies that the trained dynamics model does not memorize random noise samples in the hidden state, as described in Section III-C—if that were the case, the model would learn to overfit to each observed sample from the honeypot, which have a larger standard deviation than the spots. A random agent (Figure 3b), with actions sampled from a normal distribution, also sometimes visits the spots, and is thus capable of collecting some information about the task, but is far less efficient than the domain curious approach.

1) Bayes-optimal model: The simple nature of the environment makes it feasible to quantitatively evaluate the quality of the data provided by each method, by evaluating how much a Bayes-optimal model can learn about the task variable ξ with data provided by each method. To do this, we implemented the model in Stan [32], a Bayesian inference framework. We evaluate each model by measuring the log-probability of the true value of ξ under the ξ distribution inferred by the model n timesteps into the episode, with $n \in \{10, 20, 50, 100, 200, 500\}$.

The results of this evaluation are shown in Figure 4. We observe that the domain curious approach provides data that allows to identify the task variable accurately and with high confidence, compared to other approaches. A standard curious policy, which learns to go to the honeypot (as shown in Figure 3c), never collects any useful data about the environment, as it is attracted by environment that is highly unpredictable. A random policy gains more and more confidence as it keeps exploring, but does not get close to the results achieved by the proposed domain curiosity approach within a single episode. Notably, a random policy with standard deviation of 10 performs the best out of all tested values. Smaller values, such as 5, tend to stay close to the center or explore only one spot, whereas larger values, such as 20, tend to get stuck along the edges of the environment; a value of 10 provides a tradeoff between both behaviors.

2) ξ identification: To demonstrate how reliably the task variables can be predicted given a set of collected transitions
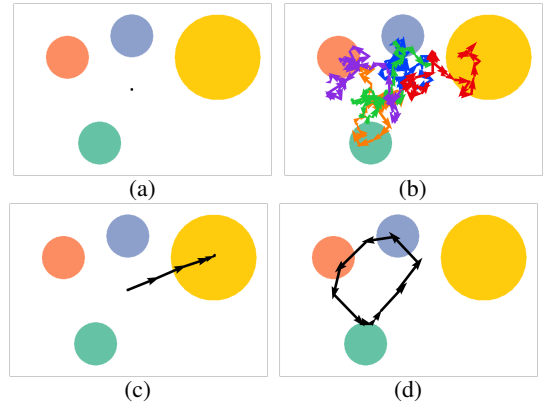

Fig. 3: The SpotsAndHoneypots environment (a) and trajectories taken by each policy during exploration: 5 rollouts of a random policy (b), a standard curious policy (c), and the proposed method (d).
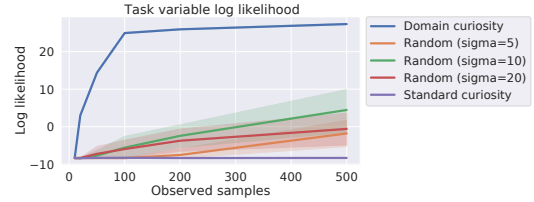

Fig. 4: Task variable log likelihood in SpotsAndHoneypots, as inferred by a Bayes-optimal model with data collected by each method

we train a recurrent neural network to predict the value of the task variable. The neural network used for these experiments is built up of three fully connected layers with ReLU nonlinearities, followed by an LSTM cell, and by three more fully connected layers with 64 neurons each. The hidden state of the LSTM is 8-dimensional.

As an additional baseline, we also include the *Uniform* dataset, which is collected by placing the agent in a uniform place in the environment, and taking a random action. This dataset has no time consistency between samples and thus cannot be realistically collected by any policy; however, it does not have any bias towards the starting position, like all the datasets collected by all tested policies do. For the random policy, in this experiment, we report values for the standard deviation of 10, as—just like in the previous experiment—it is the one that performed best out of the options we tried (1, 5, 10, and 20).

The results of this evaluation are presented in the first column of Table I. We see that, similarly to the results obtained from the Bayes optimal model, the domain curious model allows for the most accurate estimation of the dynamics parameters, with the Uniform data distribution ranking second, and the standard curious model performing worst (i.e, not learning anything about the environment).

3) Next state prediction: In addition to task variable identification, we also asses the state prediction accuracy of an adaptable dynamics model, using data collected by each of the methods. The test data for all tests in SpotsAndHoneypots was sampled from the *Uniform* data distribution; thus, in this evaluation, we assess the model's ability to learn about

predicting transitions in the uniform dataset, in an off-policy fashion using data collected by each method for learning. The model we use for this purpose is a recurrent network using the LSTM cells; we chose this architecture in order to be able to identify potential bias introduced by the BKF-based architecture used for optimizing the policy (thus, we keep the two architectures different).

The results of this evaluation are presented in the first column of Table II. We can see that, with the proposed method, the model is capable of learning the most about the state transitions in the Uniform dataset, with a noticeably smaller prediction error than other methods. The remaining error—0.68—can mostly be attributed to the honeypot, where the dynamics model cannot learn to make any predictions about the α element of the state vector. Similarly to the ξ identification task, data collected by the 'standard' curious algorithm does not provide any useful data.

The experiments in SpotsAndHoneypots show that the proposed method is capable of identifying task-specific elements of the environment, while not overfitting to observation noise. This simple environment allowed us to perform evaluation using a Bayes-optimal model for ξ identification, and later to approximate this procedure with a recurrent neural network. In later experiments, we evaluate the method's capability of handling more complex robotics setups.

B. FetchSlide

Our first robotics setup is the FetchSlide environment from OpenAI Gym [33], where the goal is to slide an object to a target position on a surface with unknown friction. The state space contains the positions and velocities of the robot end-effector and the puck. The action space contains Cartesian velocity of the end-effector and a command to open or close the gripper. To randomize the environment, we randomize the friction between the puck and the surface, similarly to [31].

In this setup, we do not provide Bayes-optimal results, as it is infeasible to implement a physics simulation engine inside a Bayesian inference framework.

1) Dynamics identification: For this evaluation, we use the same method as was used for the ξ identification in SpotsAndHoneypots. The model is trained to predict friction, given a sequence of state transitions from a single episode collected by each of the methods. The model is regularized by adding a small Gaussian noise $\epsilon \sim \mathcal{N}(0, 4 \cdot 10^{-4})$ to the position and velocity measurements used as the state. The results of this evaluation are presented in Table I.

Based on the results, we see that both the curious and the domain curious algorithm learned to provide more or less equally informative data. This can be attributed to the lack of randomness in the environment; essentially, the only thing the feedforward dynamics model can be inaccurate about is the motion of the puck; thus, domain curiosity and the standard curiosity approach both learn to hit the puck in a way that allows for dynamics identification. Interestingly, most curiosity methods aimed at aiding exploration, such as [18], would actually learn to ignore this source of randomness, as it would be treated as environment noise and ignored

by the model in the feature space; thus, for exploration tasks related to domain identification, the usual benefits of this method (apparent in single-domain exploration) would actually have an adverse effect in exploration related to domain identification.

The performance of both random policies and the standard curious algorithm in this setup is also aided by the ease of exploration in this environment—the puck is always initialized within 10cm distance from gripper, which makes this source of model error easy to find for the curious algorithm.

We also see that both curious methods perform slightly better than the task policy. This could be caused by the puck moving noticeably faster in both methods, which makes the changes between consecutive states larger, making it easier to identify changes in position and velocity, and thus improving the model's ability to estimate the dynamics.

2) Next state prediction: Similarly to FetchSlide, we also evaluate the model's ability to predict state transitions given data collected by each of the approaches. The reference policy—the one whose state transitions are being predicted—is, in this case, the task policy, which slides the puck to the goal position. In this experiment, we only predict the motion of the puck, and ignore other elements of the state space (such as joint positions), as they are not affected by the friction in any way. Thus, in this experiment we assess how well an adaptable, LSTM-based dynamics model can predict the movement of the puck, as caused by a trained task policy, given data collected from each exploration method.

The results summarized in Tab. II show that both the domain curious and standard curious approaches allow for significantly smaller prediction than a random policy, and slightly smaller than the task policy. There are no significant differences between the two curiosity approaches.

C. Box

In this setup, the Franka Panda robot is tasked with exploring the environment to locate the position of a heavy box. The position of the box is randomly selected in range of 50–70 cm along the x axis (facing along the table) and -40–40cm along the y axis (spanning the whole range between the left and right edge of the table). The state space includes the current joint position and velocities of the arm (and does not include the position of the box). The action space contains desired accelerations of the robot, making it relatively easy to enforce torque, acceleration and velocity limits of a physical robot arm. The robot is controlled by a compliant, low-gain impedance controller. The position of the box can be identified by coming in contact with and pushing onto it—due to large mass of the box and low gains of the impedance controller, it is difficult for the robot to move the box. This feedback signal, as expressed by the difference between where the robot *expected* to move and where it *actually* moved, can be used to identify the location of the box.

1) Learned motions: This policy learns sweeping motions, covering most of the space between the edges of the table, and trying to follow the edges of the box when it is encountered. The specific motion depends on the random seed,
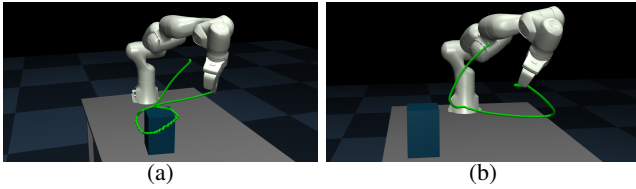
Fig. 5: Example end-effector paths (marked in green) learned by the policy in the Box environment

with a few example paths shown in Figure 5. In Figure 5a, the policy encountered the box during the sweeping motion and started to move the end-effector around it. Figure 5b shows a failure case, where the box is located too close to the edge of the table, outside of the explored area.

In contrast to the previous setup, the regular curiosity baseline did not learn any useful behaviour—it learned to rotate the joints in a constant direction until reaching joint limits (with the direction of motion varying between different random seeds). This is most likely caused by more complex exploration required to find the uncertainty in the environment—unlike the previous setup, the box is located farther away from the starting position and it is harder to find during exploration.

As an additional baseline, we used a reaching policy that moves the arm to a random place in the range where the box may be located.

2) ξ identification in simulation: Similarly to previous setups, we train a recurrent model to predict the position of the box given data collected by each baseline policy. The box position estimation errors (in meters) are presented in Table I. We observe that, similarly to previous experiments, the proposed method allows for most accurate estimation of dynamics parameters of all evaluated methods. The proposed method allows for most accurate estimation of the box position, with an average error of 3 cm. Other methods either produce an error of 21cm, which is a result of predicting the center of the range.

3) State prediction in simulation: Similarly to previous setups, we also evaluate the model's ability to learn about state transitions in data collected a reference policy, using data collected with each method. The reference policy is a policy that reaches the inside of the box (knowing its ground-truth location).

The results are shown in the third column of Table II. The prediction error is, at first, the same for all three methods; however, between 50–150 samples into the episode (the time it takes the robot to find the box), it starts to drop for the domain curious approach, reaching a slightly lower final value. For other approaches, there is no noticeable learning

4) Real-world ξ identification: To verify the method's capability to identify the task variable in real-world conditions, we constructed an analogous setup with a real-world Franka Panda robot arm. The setup is shown in Figure 6.

In this setup, we have compared the performance of the proposed method to the random reaching baseline. The baseline was selected because it was the best performing simulated baseline and because—unlike the random and curious policies—it is safe to run on physical hardware. We collected

Fig. 6: The real world FindBox setup with Franka Panda.

a total 382 trajectories from 26 different box locations for the proposed method and a total of 506 trajectories for the same box positions. Since the number of rollouts can be different for each evaluation point, each sequence in the loss term was weighted to correct for this dataset imbalance. Since the real-world table has a width of 80cm, and the size of the box is 20x20cm, we only tested x positions in the range of -30 – 30cm (to reproduce the range used in simulation, it would be required to have half of the box in the air unsupported by the table, which would likely result in the box being dropped when touched by the robot). We then split each dataset, leaving 60% of the trajectories for training and 40% for the test set. In order to prevent overfitting, the trained network was heavily regularized—we used weight decay value of 10^{-5} and injected random noise with a standard deviation of 10^{-2} into the data. We also used a smaller neural network, with two hidden layers with the size of 16.

The results of this evaluation, in the form of prediction error t timesteps into the episode, are shown in Figure 7 (where each timestep is 20ms). Similarly to the simulated results, the proposed approach allows for accurate identification of the location of the box, reaching much smaller position estimation errors than the random reaching baseline. By analyzing the shape of the domain curious curve, we see that at first, the uncertainty about the position of the box is quite large at around 17cm. After about 25 timesteps (0.5 seconds), when the robot starts the sweeping motion, the uncertainty starts to fall down, until it finally settles at around 6 cm around 175 (3.5 seconds) timesteps into the episode. This error is a bit larger than what was observed in simulation (3cm), but is still a noticeable improvement over the random reaching baseline policy.

For the random reaching policy, the error starts at more or less the same value of 16cm, and, on average, it slightly improves around 40–80 timesteps (0.8–1.6 seconds), which roughly corresponds to the time it takes the robot to reach the goal position. This indicates that the robot was able to make at least some predictions about the location of the box given the state transitions collected by the reaching policy, but these transitions were far less informative than data collected by the proposed domain curious approach.

V. CONCLUSIONS

In this paper, we presented a method for training a *domain curious policy*—a policy specifically optimized to maximize the agent's information gain about the environment. We demonstrated that the method focuses on the domain-specific, learnable aspects of the environment while ignoring
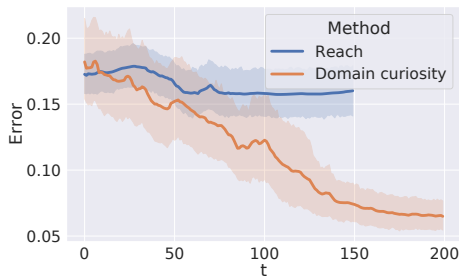
Fig. 7: Box position estimation errors t timesteps into an episode for the real-world Box setup with the proposed method and the reaching baseline

pure noise, unlike standard curiosity-based methods. We also demonstrated that the learned policies can be used to successfully learn about the unknown aspects of the environment, both in simulation and on a real robot setup.

In the proposed method, the policy is only conditioned on the current state, and has no direct measure of how much the agent has already learned. In simple environments, a successful policy can still be learned by correlating knowledge with other elements of the state space, such as position, with heuristics like *if the agent is moving towards the orange spot, it hasn't visited the spot yet*. Extending the proposed method to more complex situations may require conditioning the policy on the current task variable and its uncertainty.

To apply the method in multi-task learning scenarios, such as walking to an unknown target, a similarly structured reward prediction model could be used. The method was studied in the context of dynamics estimation, and it was assumed that only the state transition probabilities depend on ξ. The reward however remains the same across all conditions and it is thus unclear how well the proposed method would perform in such a task.

Ensuring safety is crucial in real-world applications. Thus, it is essential that the trained policy explores the environments without putting the agent itself or other actors in the environment at risk [34]. While we have provided a simple measure of safety by penalizing contact forces between the robot and other objects in the environment, engineering rewards that assure safety is difficult in general. While the measure is feasible for cases where acceptable contacts are easy to specify such as pushing, this approach would scale poorly to more complex scenarios, such as compliant interaction between a human and a robot. One possible way to address this challenge would be to learn safety measures in the simulation, together with the policy and the dynamics model. This would open a way to increase the efficiency and reliability of sim-to-real transfer by curious domain adaptation, and push sim-to-real methods toward practical applications.

References

[1] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.," in *ICML*, 2018.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML*, 2015.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[5] OpenAI, "Learning dexterous in-hand manipulation," *IJRR*, vol. 39, 2020.

[6] E. Valassakis, Z. Ding, and E. Johns, "Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics," in *IROS*, 2020.

[7] M. Hazara and V. Kyrki, "Transferring generalizable motor primitives from simulation to real world," *IEEE RA-L*, vol. 4, 2019.

[8] F. Muratore, M. Gienger, and J. Peters, "Assessing transferability from simulation to reality for reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[9] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *ICRA*, 2019.

[10] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson, "Varibad: A very good method for bayes-adaptive deep rl via meta-learning," in *ICLR*, 2020.

[11] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *ICLR*, 2019.

[12] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta reinforcement learning for sim-to-real domain adaptation," in *ICRA*, 2019.

[13] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *CoRL*, 2020.

[14] J. Schmidhuber, "A possibility for implementing curiosity and boredom in model-building neural controllers," in *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, 1991.

[15] J. Schmidhuber, "Adaptive confidence and adaptive curiosity," tech. rep., Institut fur Informatik, Technische Universitat Munchen, 1991.

[16] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE Transactions on Autonomous Mental Development*, 2010.

[17] R. Houthooft, X. Chen, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *NIPS*, vol. 29, 2016.

[18] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," ICML'17, 2017.

[19] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," in *ICML*, 2019.

[20] P. Shyam, W. Jaśkowski, and F. Gomez, "Model-based active exploration," in *ICML*, 2019.

[21] P. Morere and F. Ramos, "Bayesian rl for goal-only rewards," in *CoRL*, 2018.

[22] P. Morere, R. Marchant, and F. Ramos, "Sequential bayesian optimization as a pomdp for environment monitoring with uavs," 2017.

[23] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[24] B. Stadie, G. Yang, R. Houthooft, P. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, "The importance of sampling in meta-reinforcement learning," in *NIPS*, 2018.

[25] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," in *ICLR*, 2019.

[26] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," in *NIPS*, 2018.

[27] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng, "Stein variational policy gradient," 2017.

[28] J. Zhang, J. Wang, H. Hu, Y. Chen, C. Fan, and C. Zhang, "Learn to effectively explore in context-based meta-rl," in *Arxiv*, 2020.

[29] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Arxiv*, 2019.

[30] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *ICANN*, 2001.

[31] K. Arndt, A. Ghadirzadeh, M. Hazara, and V. Kyrki, "Few-shot model-based adaptation in noisy conditions," *IEEE RA-L*, 2021.

[32] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *Journal of statistical software*, 2017.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[34] S. H. Huang, M. Zambelli, J. Kay, M. F. Martins, Y. Tassa, P. Pilarski, and R. Hadsell, "Learning gentle object manipulation with curiosity-driven deep reinforcement learning," *ArXiv*, 2019.