Learning Active Task-Oriented Exploration Policies for Bridging the Sim-to-Real Gap

Jacky Liang  
Robotics Institute  
Carnegie Mellon University  
jackyliang@cmu.edu

Saumya Saxena  
Robotics Institute  
Carnegie Mellon University  
saumyas@andrew.cmu.edu

Oliver Kroemer  
Robotics Institute  
Carnegie Mellon University  
okroemer@cmu.edu

Abstract—Training robotic policies in simulation suffers from the sim-to-real gap, as simulated dynamics can be different from real-world dynamics. Past works tackled this problem through domain randomization and online system-identification. The former is sensitive to the manually-specified training distribution of dynamics parameters and can result in behaviors that are overly conservative. The latter requires learning policies that concurrently perform the task and generate useful trajectories for system identification. In this work, we propose and analyze a framework for learning exploration policies that explicitly perform task-oriented exploration actions to identify task-relevant system parameters. These parameters are then used by model-based trajectory optimization algorithms to perform the task in the real world. We instantiate the framework in simulation with the Linear Quadratic Regulator as well as in the real world with pouring and object dragging tasks. Experiments show that task-oriented exploration helps model-based policies adapt to systems with initially unknown parameters, and it leads to better task performance than task-agnostic exploration.

I. INTRODUCTION

Reinforcement Learning (RL) is a powerful paradigm for training robots to perform complex manipulation tasks in the real world [15]. RL methods, whether model-free or model-based, often require a lot of data that is expensive to obtain with real robots. Instead, many prior works have studied how to train a task policy with simulation data. However, due to differences in simulation and real-world dynamics as well as observation models, policies trained with simulation data tend to suffer from the simulation-to-reality gap, i.e., the distributional differences between training (simulation) and testing (real-world) data are sufficiently large to degrade the performance of the policy.

Many methods have been proposed to address the sim-to-real gap, including domain adaptation [5] and domain randomization for model-free RL [2], and learning residual models that correct sim-to-real errors for model-based RL [3]. Past works also showed that it is possible to adapt simulation parameters with real-world observations to train model-free RL policies [6], and to use models learned from real-world data to directly perform trajectory optimization for manipulation tasks [22].

If a known model with initially unknown parameters is given, System Identification (Sys-Id) can be used to tune these parameters to match the model with different instances of real-world environments. Sys-Id can be passive or active—the former uses offline trajectories or ones incurred during task execution, and the latter uses an explicit information-gathering exploration policy to probe the environment. The observation trajectories generated by such an exploration policy are used to fit the unknown parameters of the dynamics model, i.e. the simulator. Then, a model-based trajectory optimization or planning algorithm uses the model with the estimated parameters to produce a task policy, which is then executed in the real world.

Our proposed approach (Figure 1) follows this route. It explicitly learns an exploration policy that interacts with the real world and identifies the initially unknown parameters of a known model, such that task policies planned with that model can succeed in the real world. For example, the model can be a full dynamics simulation, with the unknown parameters being the mass and friction of the objects involved in the task.

The exploration policies in prior works optimize for model parameter accuracy [3] or model prediction errors [38], and as such we call them Active Task-Agnostic exploration. However, it is usually the case that some parameters are more important to trajectory optimization and task performance than others. Hence exploration policies that identify all parameters equally well may lead to worse task performance than ones that focus on parameters about which the task is sensitive.

In this work, we propose learning Active Task-Oriented exploration policies, where the exploration policy directly optimizes for the performance of the downstream task. The
exploration policy is trained in simulation over a distribution of physics and task parameters. Once learned, the exploration policy can be applied in environments with different parameters and different task instances without retraining. This is in contrast to previous works that adapt simulations to or learn models of a specific instance of real-world dynamics and task parameters, which limit such generalizations.

We performed three experiments to evaluate our framework—one in simulation with the Linear Quadratic Regulator (LQR) and two in real world with pouring and box dragging tasks. The experiments show that task-oriented exploration helps model-based policies adapt by identifying system parameters and that task-oriented exploration leads to better task performance than task-agnostic exploration. See videos and supplementary materials at https://sites.google.com/view/task-oriented-exploration/

II. RELATED WORKS

One popular method to address the sim-to-real gap is Domain Randomization (DR), which can be applied for observation models [30, 32] or dynamics models [25, 21, 19, 20, 28], or both [2]. DR does not aim to train a policy with simulation parameters that is close to those of the real world. Instead, DR trains the policy with a wide distribution of parameters, with the idea that a policy that can perform the task under most of the different simulations should also be able to perform it in the real world. As such, DR can make policies more robust. However, training a policy that works on average for all parameters may lead to sub-optimal performance when different parameters require different policy behaviors [38, 19]. In addition, DR often needs humans to fine-tune the parameter distribution with some prior knowledge about the appropriate range of parameter values in the real world. The authors of [19, 2] address this issue by effectively building a curriculum that actively changes the DR distributions during training, leading to better policy generalization than sampling from a static and uniform parameter distribution. The authors of [6, 28] adapt the parameter distribution to better match real-world observations to train model-free RL policies. However, such adaptation is specific to one instance of real-world environments, so the trained policy is not expected to generalize on environments with different physics or task parameters.

Numerous works have studied using predefined trajectories for Sys-Id [24], especially in the context of identifying contact dynamics [35, 14, 9]. It is also possible to perform manual Sys-Id by directly measuring dynamics parameters in the real world [31]. However, these methods do not easily scale to different kinds of robots and tasks. Alternatively, prior works have explored finding the physics parameters, whether for model-free policy learning or model-based adaptations, through trajectories incurred by the task policy. We refer to these as Passive Sys-Id methods. For example, the authors of [36] train a physics parameter-conditioned model-free task policy and a separate prediction model to predict the parameters from a history of trajectories generated by the task policy. The authors of [27] train an environment-embedding-conditioned RL agent that can identify the embedding online during task execution. On the model-based side, the authors of [29] propose a method to iteratively learn the model in an on-line fashion with trajectories generated by an optimal controller that uses the model, and the algorithm in [16] learns a dynamics model conditioned on a local context embedding, extracted by a learned context encoder during task execution. Passive Sys-Id has the limitation that the trajectory generated by the task policy might not be the most suitable ones for identifying model parameters.

Instead of finding real-world model parameters to improve model accuracy, many works have also studied using real-world data to directly learn the model from scratch or corrections on top of a known but imperfect model. An example of the former is [22], where the authors learn a dynamics model for in-hand manipulation tasks using real-world interactions and perform the task with trajectory optimization. Works doing the latter are commonly referred to as residual learning [15, 1, 10, 37]. The method in [3] combines this idea with Passive Sys-Id to produce a model that can iteratively reduce the residual errors with little real-world data.

By contrast to Passive Sys-Id, Active Sys-Id algorithms interact with the environment to explicitly identify relevant system parameters. This is common for Interactive Perception, where a robot performs probing actions in an environment to segment objects [33], infer object properties like mass [12], or infer kinematical constraints [11, 14, 7]. Like our approach, many of these works assume a known model with unknown parameters (e.g. the type of joint connecting two rigid bodies), and they aim to choose the most informative actions via heuristics like maximum information gain to reduce dynamics prediction error. The work in [38] learns an exploration policy that generates informative trajectories for inferring environment embeddings. These embeddings are used as input to model-free RL policies, and the embedding encoder is trained to reduce dynamics prediction error. In [23] the authors train an exploration policy that optimizes for the accuracy of a parameter prediction network that estimates the mass of articulated objects. We describe Active Sys-Id methods like these as Task-Agnostic, because the exploration is done to optimize for the accuracy of all of the model parameters or model predictions, and not a downstream task.

Our work considers the case for performing Active and Task-Oriented exploration, where the exploration policy is used to infer model parameters in a way that directly optimizes for task performance. The algorithm proposed in [18] is similar—it applies a Bandits-based approach to select from a fixed set of experiments to perform in the environment. Like [38] however, this work focuses on the case of a finite set of environments, so the exploration policy only needs to produce information that can classify which environment the agent is in, and not the underlying system parameters. The authors of [8] propose learning a model from scratch that minimizes prediction error in the value function, thus making model-learning task-oriented. In this work, we assume
a known parameterized model and explicitly try to identify
the continuous task-relevant system parameters. This means
that the method does not need a predefined discrete set of
environments. It also means the method does not require policy
learning or model learning from scratch, because the tuned
model is directly used for trajectory optimization to perform the
downstream task.

III. M ETHOD O VERVIEW

We first give a general overview of the proposed framework,
seen in Figure 2 for learning active and task-oriented exploration
policies. Following this section are three instantiations of
the framework—one simulated Linear Quadratic Regulator
(LQR) task and two real-world manipulation tasks.

A. Active and Task-Oriented Exploration

We consider the problem of fitting the parameters of a
dynamics model by using an exploration policy to generate
interactions with the real system, so that a task policy planned
with the fitted dynamics model can be directly applied on the
real system with no further fine-tuning.

Our chief assumption is that for a given task there exists
some values of model parameters that will make the simul-
ated dynamics sufficiently match real-world dynamics, so a
task policy planned using the estimated parameters can also
complete the task in the real world. We further assume that it is
possible to recover these parameter values from observations
available to the robot, and that we have models of the objects,
the robot, as well as their initial states.

We define the following variables:
• \( x_t \) - state (e.g. robot, object configurations).
• \( u_t \) - robot actions (e.g. desired end-effector poses)
• \( o_t \) - observations (e.g. tracked object trajectories)
• \( \theta \) - dynamics model parameters that can affect task
  execution (e.g. mass, friction).
• \( f_\theta(x_t, u_t) \) - a discrete-time dynamics function that yields
  \( x_{t+1} \) using the parameters \( \theta \). This is used to describe
  both simulated and real-world dynamics functions. The
  simulated dynamics is deterministic, while real-world
dynamics is stochastic due to unmodeled noise in the
environment.

Model-based Task Policy. Let \( \pi_e(\theta) \) be a task policy
that uses the dynamics model with model parameters \( \theta \). For
example, it could be a policy that performs a motion plan
generated by trajectory optimization with a physics simulator
as the dynamics model. Let \( T_e \) be the time horizon of the task.
Forming \( \pi_e \) is called Trajectory Optimization (TrajOpt).

Let \( J(\pi_e(\theta), \theta) \) be the total cost of a task performed by
a policy based on \( \theta \) acting in an environment that actually has
physics parameters \( \theta \). \( J \) can be an expectation over a
distribution of tasks (e.g. a distribution over goal states), but
for brevity we omit the expectation symbol \( \mathbb{E} \) when writing
\( J \). The expected loss over \( \theta \) is \( \mathbb{E}_\theta J(\pi_e(\theta), \theta) \). If the policy is
optimized for one set of parameters but deployed on a system
that has different parameters, we expect the performance of

\[
\hat{\theta} = \arg\min_{\theta} \mathbb{E}_\theta J(\pi_e(\theta), \theta)
\]

Algorithm 1 Deploy Exploration and Task Policy

Input: \( \pi_e, f_\theta \)
1: Roll out \( \pi_e \) in environment with dynamics \( f_\theta \)
2: Obtain exploration trajectory \([x_0, o_0:T_e, u_0:T_e]\)
3: Estimate model parameters \( \hat{\theta} \leftarrow g(x_0, o_0:T_e, u_0:T_e) \)
4: Form \( \pi_\tau(\hat{\theta}) \) via trajectory optimization using \( f_\theta \)
5: Roll out \( \pi_\tau \) in \( f_\theta \)
6: return \( J(\pi_\tau(\hat{\theta}), \theta) \)

the policy to be worse than one with the parameters it was
optimized for: \( \mathbb{E}_\theta J(\pi_e(\theta), \theta) \geq \mathbb{E}_\theta J(\pi_e(\hat{\theta}), \hat{\theta}) \).

Simulation Parameter Optimization. An exploration trajectory
consists of an initial state \( x_0 \), real-world observations \( o_0:T_e \), and actions \( u_0:T_e \), where \( T_e \) is the horizon for the
exploration trajectory. Let \( g \) be the optimizer that optimizes
for the physics parameters that match these trajectories:
g(\( x_0, o_0:T_e, u_0:T_e \)) = \theta. We call applying \( g \) Simulation Op-
timization (SimOpt). For example, \( g \) can be a closed-form
expression that directly solves for \( \theta \) or a derivative-free op-
timization algorithm that iteratively searches for it. In general, \( g \)
tries to give an estimate \( \hat{\theta} \) that minimizes the prediction error
of the resultant dynamics model on the exploration trajectory.

Active Task-oriented Exploration Policy. Let \( \pi_e \) be the
task-oriented exploration policy, which acts as a feedback
controller starting from an initial state that is given or learned.
The exploration trajectory \( \{u_0:T_e, o_0:T_e\} \) generated by \( \pi_e \) is
given to \( g \), yielding estimated model parameters \( \hat{\theta} \) that are
used by the task policy \( \pi_\tau(\hat{\theta}) \) to perform the task. We call
this the \( \pi_e \rightarrow \pi_\tau \) pipeline.

The goal of \( \pi_e \) is to generate an exploration trajectory that
leads to low expected costs incurred by \( \pi_e \):
\[
\pi_e = \arg\min_{\pi_e} \mathbb{E}_\theta \left[ \Psi(\pi_e(\theta), \theta) + \gamma h(\pi_e) \right]
\]

where \( h(\pi_e) \) is a regularization term that penalizes \( \pi_e \) from
being too complex and incurring states and actions that have
high costs, and \( \Psi \) denotes the regret of \( \pi_\tau(\theta^*) \) w.r.t. a task
policy optimized using the real dynamics parameters:
\[
\Psi(\pi_\tau(\hat{\theta}), \theta) = J(\pi_\tau(\hat{\theta}), \theta) - J(\pi_\tau(\theta), \theta)
\]

See Algorithm 1 for the function that deploys the \( \pi_e \rightarrow \pi_\tau \)
pipeline at “test time” to perform a task. The deploy func-
tion takes in a dynamics model \( f_\theta \), which is the real-world
dynamics when the robot is actually performing the task and
the simulated dynamics during training.

This procedure is active, because it chooses how to interact
with the real system to generate informative observations
for estimating model parameters. It is also task-oriented as
opposed to task-agnostic, because \( \pi_e \)’s goal is to minimize
task regret \( \Psi \), and not the accuracy of \( g \)’s predictions.

B. Training a Task-Oriented Exploration Policy

The exploration policy \( \pi_e \) is trained in simulation via RL,
where the reward function is the negative of the objective in
Equation 1. At each training iteration we sample “ground-
truth” physics parameters from a wide, predefined distribution
\( \theta \sim \Theta \). Then we go through the \( \pi_e \rightarrow \pi_\tau \) pipeline, deploy
Algorithm 2 Evaluate Expected Regret of Exploration Policy

Input: \( \pi_e, \Theta, N \)

1: for \( n \in \{1 \ldots N\} \) do
2: Sample \( \theta_n \sim \Theta \)
3: Form \( \pi_\tau(\theta_n) \) via trajectory optimization using \( f_\theta \)
4: Evaluate \( J(\pi_\tau(\theta_n), \theta_n) \)
5: \( J(\pi_e(\hat{\theta_n}), \theta_n) \leftarrow \text{Deploy}(\pi_e, f_{\theta_n}) \)
6: \( \Psi_n \leftarrow J(\pi_\tau(\theta_n), \theta_n) - J(\pi_e(\hat{\theta_n}), \theta_n) \)
7: end for
8: return \( \mathbb{E}_\theta \Psi \approx \frac{1}{N} \sum_{n=1}^N \Psi_n \)

the task policy \( \pi_\tau \) in the “ground-truth” simulation \( f_\theta \), and evaluate the task regret \( \Psi \). If there is a distribution of task objectives (e.g., goal object poses), we repeat this process with multiple task samples. This constitutes one rollout of \( \pi_e \). The reward for \( \pi_e \) is sparse, as it can only be computed after the entire exploration trajectory is performed. See Algorithm 2 for how to evaluate the objective’s expected regret term, \( \mathbb{E}_\theta \Psi(\pi_\tau(\theta), \theta) \), in simulation during training.

Sampling \( \theta \) from a distribution, instead of training \( \pi_e \) for a specific \( \theta \), is motivated by works in DR. We apply DR’s argument not to the task policy, but to the exploration policy; \( \pi_e \) should be applicable to a wide range of simulated environments, and if it is, then it should also be able to work in the real world even if it was trained in simulation. This should be easier to achieve than with the task policy, because the space of trajectories that completes a task is more restrictive than one that’s informative for Sys-Ild.

Optimizing for the expected task regret \( \mathbb{E}_\theta \Psi(\pi_\tau(\theta), \theta) \) w.r.t. \( \pi_e \) is equivalent to optimizing for the expected task performance \( \mathbb{E}_\theta J(\pi_e(\theta), \theta) \). This is because the the difference, the expected task performance of the policy using ground-truth system parameters \( \mathbb{E}_\theta J(\pi_\tau(\theta), \theta) \), does not depend on the exploration policy. In practice, we optimize for \( \mathbb{E}_\theta J(\pi_e(\theta), \theta) \), but during training we report \( \mathbb{E}_\theta \Psi(\pi_\tau(\theta), \theta) \), because \( \Psi \) fluctuates less than \( J \) with noisy dynamics, and \( \Psi = 0 \) is an interpretable target for training \( \pi_e \).

IV. ANALYSIS AND EXPERIMENTS FOR LQR

To better understand the proposed framework, we first apply it to the LQR task, which is fast to train and amenable to analysis. The task is to form a linear feedback controller to act in a fully observable discrete-time linear system to minimize a finite-horizon quadratic cost in terms of the states and actions:

\[
    x_{t+1} = Ax_t + Bu_t
\]

where \( x_t \in \mathbb{R}^n \), \( u_t \in \mathbb{R}^m \). This is an interpretable target for training with noisy dynamics, and \( \Psi = 0 \) is an interpretable target for training \( \pi_e \).

A. LQR Analysis

Our goal in this section is to rewrite the minimization of the task cost \( \min J \) as an optimization problem w.r.t to an exploration policy \( \pi_e \) and see how the task-oriented objective affects the optimization. In the equations that follow, we denote \( A \) as the ground-truth system dynamics, and \( \hat{A} \) as the estimated system dynamics. The cost \( J \) is computed with respect to \( A \), while the model-based LQR policy \( \pi_\tau \) is computed with respect to \( \hat{A} \). We assume \( B \) is known.

**Trajectory Optimization.** With the task of minimizing \( J \), the optimal LQR policy is a linear feedback controller with time-varying gains \( u_t = K_{\tau,t}x_t \), where the gains are computed as follows:

\[
    P_{\tau,t} = Q
\]

\[
    K_{\tau,t} = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} \hat{A}
\]

\[
    P_t = Q + K_{\tau,t}^T R K_{\tau,t} + (\hat{A} + BK_{\tau,t})^T P_{t+1} (\hat{A} + BK_{\tau,t})
\]

**Simulation Optimization.** With linear dynamics and full observability, we can derive the closed form solution for \( \hat{\theta} \) given an exploration trajectory by using the following objective:

\[
    \hat{\theta} = g(x_{0:T_e}, u_{0:T_e}) = \arg\min_{\theta} \sum_{t=1}^{T_e} \| \hat{x}_t - x_t \|_2^2
\]
Taking the derivative and setting it equal to 0 yields:

\[
\dot{\theta} = \arg \min_{\theta} \sum_{t=1}^{T_e} \|U^T x_{t-1} + Bu_{t-1} - x_t\|^2_2
\]

\[
= -U^T \left( \sum_{t=1}^{T_e} x_t x_t^T \right) U^T \left( \sum_{t=1}^{T_e} (U^T x_t) \circ (U^T (Bu_{t-1} - x_t)) \right)
\]

where \( \circ \) denotes element-wise product and \( \dagger \) the pseudo-inverse. If \( T_e \geq n \) and \( \{x_1 \ldots x_{T_e}\} \) spans \( \mathbb{R}^n \), then \( \sum_{t=1}^{T_e} x_t x_t^T \) is invertible. In practice, we use the pseudoinverse to handle edge cases and also resolve numerical instabilities.

**Task-Oriented Exploration Policy.** We set the exploration policy to be a linear feedback controller with time-invariant gains: \( u_t = K_c x_t \). Under the proposed framework, \( \pi_e \) will generate a trajectory of length \( T_e \), which will be used to estimate \( \hat{A} = U \theta U^T \). The estimated dynamics model is used to compute the optimal LQR gains, the performance of which will be evaluated in \( J \).

To make the analysis tractable, we make the following assumptions: \( n = m, B = I, Q \) is diagonal, and \( R = 0 \). Setting \( R = 0 \) is the strongest assumption, and it is not met in practice, as doing so gives zero penalties to arbitrarily large controller effort (although the closed loop system does not converge in 1 step if there are modeling errors \( \|\hat{A} - A\| > 0 \)). However, having \( R = 0 \) greatly simplifies the algebra that follows, and the result still provides useful insights.

With these assumptions the LQR equations yield \( P_{T_e} = Q, K_t = \hat{A}, P_T = Q \). We can rewrite the costs in terms of the closed-loop dynamics by using this simplified LQR policy:

\[
x_{t+1} = (A + BK_t)x_t = (A - \hat{A})x_t
\]

\[
J = \sum_{t=1}^{T_e} x_t^T Q x_t = \sum_{t=1}^{T_e} x_{t-1}^T (A - \hat{A})^T Q (A - \hat{A}) x_{t-1}
\]

Finding the optimal \( K_c \) to minimize \( J \) can be formulated as an optimization problem on setting the gradient \( \nabla_K J \) to 0:

\[
K_c^* = \arg \min_{K_c} \|\nabla_K J\|
\]

\[
= \arg \min_{K_c} \|\nabla_K \dot{\theta} \nabla_\theta J\|
\]

\[
= \arg \min_{K_c} \|\nabla_K \dot{\theta} \sum_{t=1}^{T_e} U^T (D_t + D_t^T) U\|
\]

where \( D_t = x_{t-1} x_{t-1}^T (A - \hat{A}) Q = x_{t-1} (\hat{x}_t - x_t)^T Q \).

Note that the objective function of optimizing the task cost \( J \) w.r.t. the exploration policy \( K_c \) is a combination of 1) how sensitive the identified parameters \( \dot{\theta} \) are to the exploration policy (\( \nabla_K \dot{\theta} \)), 2) the dynamics prediction error (the \( \hat{x}_t - x_t \) term of \( D_t \) weighted by 3) the task costs (the \( Q \) term of \( D_t \)). As \( R_c \) depends on all 3 of these factors, this analysis on the simplified system illustrates the difference between task-oriented system identification vs. the task-agnostic variant, which would not have terms that depend on task performance.

![Fig. 3. Comparison of task-agnostic vs. task-oriented exploration policy training in regret ratio of LQR cost on test systems across 10 random seeds. Task-oriented exploration achieves lower final regret ratio, converges faster, and incurs lower variance on task regret than task-agnostic exploration.](image-url)
dynamics parameters in this case would not be useful for downstream tasks.

Figure 3 plots the results of this experiment. The x-axis denotes the number of training batches. The y-axis denotes the ratio between the regret achieved by the exploration policy on test systems vs. the regret of the initial random exploration policy, which is the same for both task-agnostic and task-aware training runs. Regret ratio is reported here, because the unitless LQR cost is difficult to interpret, and we can compute the optimal regret and provide a more intuitive value between 0 and 1. We ran the training procedure for 10 random seeds, and the means and standard deviations are computed across those seeds.

While both task-agnostic and task-oriented exploration policies are able to reduce regret, the task-oriented exploration performs better than the task-agnostic variant by having faster convergence toward a lower final regret ratio, as well as having a smaller variance.

V. REAL-WORLD ROBOT EXPERIMENTS

We apply the framework to two real-world robot manipulation tasks, one using an analytical model and a discrete exploration action space, and one using full dynamics simulations with continuous exploration action space.

A. Task: Pouring

In the pouring task, the robot must pour $m_r$ kg of water from a cup with known shape but containing initially unknown amount of water. The goal parameter $m_r$ is sampled at every execution of the task: $m_r \sim N(m_r, \sigma_{m_r})$. Because the cup shapes are known a priori, if we know the initial amount of water in each cup, we can compute the correct angle at which to tilt the cup to achieve the desired amount $m_r$. The policy $\pi_r$ in this case has just one parameter—the cup tilt angle $\phi$ ($\phi = 0$ when the cup stands upright and $\phi = \pi/2$ when the cup is laying horizontal). Let $\tilde{m}_r$ refer to the actual amount of water poured. The task cost is $J = |m_r - \tilde{m}_r|$. Below is the analytical solution that relates the tilt angle $\phi$ of a cylindrical cup with uniform radius $r$, the height $h$, and the maximum volume $V$ of fluid that remains in the cup:

$$\phi = \tan^{-1}\left(\frac{1}{r}(h - \frac{V}{\pi r^2})\right) \quad (16)$$

This equation is used to compute $\phi$ given a desired $V$, calculated from the amount of water that should be left in the cup after pouring ($m_c - m_r$). Note the above model only works when $\phi < \tan^{-1}(\frac{h}{r})$, and we enforce this constraint during experiments.

In addition to the cup the task uses, another identical distractor cup is also in the scene. The unknown system parameter are the initial masses of both of the cups $\theta = [m_1, m_2]$. The exploration policy $\pi_e$ operates in a discrete action space—lifting either cup 0 or cup 1 and uses the end-effector force measurements to estimate the mass of the lifted cup. This measurement is noisy, so lifting a cup more times result in a more accurate mass measurement, which would in turn lead to smaller task costs. The exploration policy acts as follows—it first performs a single measurement for both of the cups. For the remaining $T_e - 2$ time steps, it samples which cup to lift from a Bernoulli distribution with parameter $p_e$, where a value of 1 means choosing the task-relevant cup, and 0 the task-irrelevant cup. Ideally, a trained exploration policy has a $p_e$ that strongly favors the cup the task uses, leading to a more accurate initial mass estimate and better task performance.

We trained $\pi_e$ by sampling from the analytical model with added observation noise to the mass measurements as well as dynamics noise to the outcome of how much water was poured for a given tilt angle. $\pi_e$ is optimized via gradient descent with finite-difference approximations of $\frac{\partial J}{\partial p_e}$. We set $T_e = 6$, so the maximum number of measurements per cup is 5.

Figure 4 shows both the costs incurred by the task-oriented and task-agnostic exploration policies and their $p_e$'s during training. The task-agnostic $p_e$ is around 0.6, while the task-oriented $p_e$ is closer to 0.9. The task-oriented exploration policy also achieves better final task performance.

We performed the pouring task in the real world with a 7 DoF Franka Panda arm. Two identical plastic beakers were used for the cups, and the masses before and after pouring were measured with a scale. We evaluated the trained task-oriented and task-agnostic policies with 10 samples of goal parameters in the real world. See Figure 3 for the real-world experiment setup and results. The task-oriented exploration policy, by focusing exploration on the cup more likely to be used by the task, achieves a lower average task cost of 14g instead of 22g.

B. Task: Dragging

In this task, a box object of uniform density needs to be dragged on a planar surface from an initial 3D pose (2D translation and 1D rotation $[x_0, y_0, \phi_0]$) to a target goal pose $[x_g, y_g, \phi_g]$ that is sampled from a distribution every time the task is executed. Dragging means the robot end-effector pushes the object against the workspace and drags the object along a 2D plane while maintaining contact with the object. There are 3 parameters varied and estimated in this task: the torsional friction of the robot-object contact, torsional friction of the object-table contact, and the mass of the object.

The task cost is the weighted sum of the magnitude of the
that maintains the current optimal parameters as a multivariate
use the episodic variant of Relative Entropy Policy Search
Nvidia GTX 1080 Ti GPU.
of ∆ single GPU. We simulate
which allows us to run multiple simulations in parallel on a
cally, we use the GPU-accelerated Nvidia Flex simulator [17],
and pouring tasks, here we use a physics simulator. Specifi-
the task trajectories.
Instead of using analytical dynamics models as in the LQR
and pouring tasks, here we use a physics simulator. Specifi-
cally, we use the GPU-accelerated Nvidia Flex simulator [17],
which allows us to run multiple simulations in parallel on a
single GPU. We simulate 20 robots in parallel at a time step of
Δt = 0.01s, and this achieves roughly 100 FPS on a single
Nvidia GTX 1080 Ti GPU.

1) Optimizers: For both TrajOpt and SimOpt we
use the episodic variant of Relative Entropy Policy Search
(REPS) [26]. This is a derivative-free optimization algorithm
that maintains the current optimal parameters as a multivariate
normal distribution, and it updates the mean and covariance
of the distribution at every optimization step subject to a KL-
divergence constraint.

Trajectory Optimization. The initial mean trajectory REPS
uses consists of the first waypoint right above the center of
the object, and a second waypoint to coincide with the goal
delta pose for the object. This initial trajectory doesn’t work
in most of the cases, because, depending on the friction and
mass values, the object will slip and rotate different amounts,
so its contact with the robot end-effector is not rigid. REPS
for TrajOpt converges within 10 iterations.

Simulation Optimization. Given a trajectory of object
poses and robot actions during exploration, we use REPS to
find the θ in simulation that generates the trajectory closest
to the observations. The initial mean of the dynamics model
parameters are sampled from the wide distribution Θ, while
the initial covariance is set wide enough to sufficiently cover
Θ. At every REPS iteration, sampled θ’s are used to form
the dynamics model fθ, which is then used to playback the
recorded exploration trajectory. The translation and rotational
differences between each simulation’s object poses and the
observed object poses are used to form a weighted sum cost
similar to the one in trajectory optimization. REPS for SimOpt
also converges within 10 iterations.

Training the Exploration Policy. For optimizing the ex-
ploration policy, we first experimented with REPS, but taking
the full expectation of Ψ is too slow in practice. Instead, like
with the pouring task, we use finite difference in simulation to
directly perform gradient updates and estimate the gradients by
taking a small batch of samples. To make multi-dimensional
finite difference more accurate and efficient, we sample small
perturbations around the input variable, evaluate the function
at those perturbations, and fit a plane to estimate the gradient.

Since we want the task to generalize across a distribution
of task parameters, evaluating J in Algorithms 1 and 2
requires estimating an expectation as well. To reduce the
nested sampling of J and Θ during finite difference, we split
the gradient Eθ ∇πψΨ(πx(θ), θ) into two parts via chain rule
and estimate them separately. The first term is the gradient
of the task cost w.r.t. the simulation parameters evaluated around
the estimated simulation parameters; ∇θ EθΨ(πx(θ), θ). The
second is the gradient of the estimated simulation parameters
w.r.t to the exploration policy: ∇ψ Eθ ∇πψΨ(πx(θ), θ). Gradient updates
were performed via the Adam optimizer.

Figure 9 shows the task regret ratio of task-agnostic vs. task-
oriented exploration policy during training. Similar to previous
cases, the task-oriented exploration policy achieves lower task
regret than the task-agnostic exploration policy.

2) Real-world Evaluations: To evaluate the two explo-
ration policies in the real-world, we 3D printed a box with the
exact dimensions as the one used in simulation. The box has
a cavity with a removable lid, so we can vary the box’s mass.
To vary the friction parameters of the robot-object and object-
table contacts, we attached different sheet materials to both
the top of the box and the top of the table surface. We also
attached AprilTags [34] to the top of the lid to estimate the
poses with the box, and the different materials used. In total, we experimented with the table surface.

The dragging trajectories consist of start and end 3D way points for the robot's end-effector, and they're denote by the red and green axes. While both trajectories have comparable translation and rotation magnitudes, the task-oriented exploration trajectory begins further away from the object’s center of mass. This leads to an object trajectory that is more sensitive to the torsional friction between the object and the table surface.

We evaluate each set of parameters with 3 different goal poses with 2 trials each, and we removed all trials during which the robot was not able to move the box at all. This happens when the friction of the object-table contact is much greater than that of the robot-object contact. In total, there are 48 trials used for evaluating each of the task-oriented and task-agnostic exploration policies. See Figure 8 for a visualization of the learned exploration trajectories and Figure 3 for task performance during training and real-world evaluations. The Task-Oriented exploration policy led to smaller mean and standard deviation of costs than the Task-Agnostic policy, which did not improve over random exploration. For this task, random exploration already leads to reasonable Sys-ID, and task-oriented information is needed for further improvements.

C. Discussions

In both the pouring and box dragging task, and during both training the exploration policy and testing it in the real world, the parameters identified by task-oriented exploration led to better final task performance than ones identified by task-agnostic exploration. This behavior is observed with tasks using both analytical models (LQR, pouring tasks) and black-box models (box dragging), and with discrete exploration actions (pouring tasks) and continuous ones (LQR, box dragging).

The advantage of task-oriented over task-agnostic exploration is due to the limited exploration budget. Exploration policies have a finite time horizon, and during training regularization terms are added to prevent the policy from incurring high-cost states or actions (e.g. actions or states that are too large). As a result, there is a need to explore more about system parameters to which the task cost is more sensitive. This is reflected by weighting the dynamics prediction error by the task cost in LQR (Equation 15), measuring the mass of the task-relevant cup more in pouring (Figure 4), and exploring from an initial contact further away from the object’s center of mass in box dragging (Figure 8).

Because the performance of an exploration policy during training is evaluated as an expectation over a wide distribution of system parameters as well as task goals, the learned exploration policies can be applied to different systems and generalize across the task distribution. This benefit also points to a limitation of our approach, which is that optimizing the exploration policies requires 3 layers of nested sampling. They include taking samples for finite difference approximation, system parameters, and task goals. As a result, the gradient $∇_{π, E, Ψ}$ can be slow to evaluate when full dynamics simulation is used. However, tasks with many model parameters typically only have a small subset of parameters that significantly affect task performance. As such, the benefits of the task-oriented approach will be more apparent in high-dimensional tasks, where the effective task-oriented dimenaitonalty is much lower than the task-agnostic dimensionality.

VI. Conclusion

In this paper, we proposed, analyzed, and implemented a framework of learning active task-oriented exploration policies to improve task performance in the real world and bridge the sim-to-real gap. The learned exploration policy works across system parameters and task goals, so it can be applied to different variations of the task without retraining. We instantiated the framework with three experiments using analytical and full dynamics simulation models. Across all experiments we observed that task-oriented exploration leads to better task performance than task-agnostic exploration.

ACKNOWLEDGEMENTS

The authors thank Shivam Vats, Steven Lee, and Kevin Zhang for their support in this project. This work is funded by the NSF Graduate Research Fellowship Program Grant No. DGE 1745016, NSF Award No. CMME-1925130, the Office of Naval Research Grant No. N00014-18-1-2775, and ARL grant W911NF-18-2-0218 as part of the A2I2 program.


