FLOW-BASED MAXIMUM ENTROPY DOMAIN RANDOMIZATION FOR MULTI-STEP ASSEMBLY

Anonymous authors

Paper under double-blind review

ABSTRACT

Domain randomization in reinforcement learning is an established technique for increasing the robustness of control policies learned in simulation. By randomizing properties of the environment during training, the learned policy can be conformant to uncertainty along the randomized dimensions. While the environment distribution is typically specified by hand, in this paper we investigate the problem of automatically discovering this sampling distribution via entropy-regularized reward maximization of a neural sampling distribution in the form of a normalizing flow. We show that this architecture is more flexible and results in better robustness than existing approaches to learning simple parameterized sampling distributions. We demonstrate that these policies can be used to learn robust policies for contact-rich assembly tasks. Additionally, we explore how these sampling distributions, in combination with a privileged value function, can be used for out-of-distribution detection in the context of an uncertainty-aware multi-step manipulation planner.

1 INTRODUCTION

Reinforcement learning (RL) has proven to be a useful tool in robotics for learning control or action policies for tasks and systems which are highly variable and/or analytically intractable (Luo & Li (2021); Zhu et al. (2020); Schoettler et al. (2020)). However, RL approaches can be inefficient, involving slow, minimally parallelized, and potentially unsafe data-gathering processes when performed in real environments (Kober et al. (2013)). Learning in simulation eliminates some of these problems, but introduces new issues in the form of discrepancies between the training and real-world environments (Valassakis et al. (2020)).

Successful RL from simulation hence requires efficient and accurate models of both robot and environment during the training process. But even with highly accurate geometric and dynamic simulators, the system can still be only considered partially observable (Kober et al. (2013))—material qualities, inertial properties, perception noise, contact and force sensor noise, manufacturing deviations and tolerances, and imprecision in robot calibration all add uncertainty to the model.

To improve the robustness of learned policies against sim-to-real discrepancies, it is common to employ domain randomization, varying the large set of environmental parameters inherent to a task according to a given underlying distribution (Muratore et al. (2019)). In this way, policies are trained to maximize their overall performance over a diverse set of models. These sampling distributions are typically constructed manually with Gaussian or uniform distributions on individual parameters with hand-selected variances and bounds. However, choosing appropriate distributions for each of the domain randomization parameters remains a delicate process (Josifovski et al. (2022)); too broad a distribution leads to suboptimal local minima convergence (see Figure 3), while too narrow a distribution leads to poor real-world generalization (Mozifian et al., 2019; Packer et al., 2018). Many existing methods rely on real-world rollouts from hardware experiments to estimate dynamics parameters (Chebotar et al. (2019); Ramos et al. (2019); Tiboni et al. (2023); Muratore et al. (2022)). However, for complex tasks with physical parameters that are difficult to efficiently or effectively sample, this data may be time-consuming to produce, or simply unavailable.

An ideal sampling distribution enables the policy to focus training on areas of the distribution that can feasibly be solved in order to maximize the overall success rate of the trained policy while not

wasting time on unsolvable regions of the domain. Automating updates to parameter distributions during the training process can remove the need for heuristic tuning and iterative experimentation (Mozifian et al. (2019); OpenAI et al. (2019); Tiboni et al. (2024)). In this paper, we present GoFlow, a novel approach for learned domain randomization that combines actor-critic reinforcement learning architectures (Schulman et al., 2017; Haarnoja et al., 2018) with a neural sampling distribution to learn robust policies that generalize to real-world settings. By maximizing the diversity of parameters during sampling, we actively discover environments that are challenging for the current policy but still solvable given enough training.

As proof of concept, we investigate one real-world use case: contact-rich manipulation for assembly. Assembly is a critical area of research for robotics, requiring a diverse set of high-contact interactions which often involve wide force bandwidths and unpredictable dynamic changes. Recently, sim-to-real RL has emerged as a potentially useful strategy for learning robust contact-rich policies without laborious real-world interactions Noseworthy et al. (2024); Tang et al. (2023a); Zhang et al. (2024). We build on this work by testing our method on the real-world industrial assembly task of gear insertion.

Lastly, we extend this classical gear insertion task to the setting of multi-step decision making under uncertainty and partial observability. As shown in this paper and elsewhere Tiboni et al. (2024); Mozifian et al. (2019); OpenAI et al. (2019), policies trained in simulation have an upper bound on the environmental uncertainties that they can be conformant to. For example, a visionless robot executing an insertion policy can only tolerate so much in-hand pose error. However, estimates of this uncertainty can be used to inform high level control decisions, e.g., looking closer at objects to get more accurate pose estimates or tracking objects in the hand to detect slippage. By integrating a probabilistic pose estimation model, we can use the sampling distributions learned with GoFlow as an out-of-distribution detector to determine whether the policy is expected to succeed under its current belief about the world state. If the robot has insufficient information, it can act to deliberately seek the needed information using a simple belief-space planning algorithm. For example, an in-hand camera can be used to gather a higher resolution image for more accurate pose estimation if necessary.

Our contributions are as follows: We introduce GoFlow, a novel domain randomization method that combines actor-critic reinforcement learning with a learned neural sampling distribution. We show that GoFlow outperforms fixed and other learning-based solutions to domain randomization on a suite of simulated environments. We demonstrate the efficacy of GoFlow in a real-world contact-rich manipulation task—gear insertion—and extend it to multi-step decision-making under uncertainty. By integrating a probabilistic pose estimation model, we enable the robot to actively gather additional information when needed, enhancing performance in partially observable settings.

2 Related Work

Recent developments in reinforcement learning have proven that policies trained in simulation can be effectively translated to real-world robots for contact-rich assembly tasks (Zhang et al., 2024; Tang et al., 2023a; Noseworthy et al., 2024; Jin et al., 2023). One key innovation that has contributed to the development of robust policies is domain randomization (Chen et al., 2021; Peng et al., 2017), wherein environment parameters are sampled from a distribution during training such that the learned policy can be robust to environmental uncertainty on deployment.

Some previously explored learning strategies include minimization of divergence with a target sampling distribution using multivariate Gaussians (Mozifian et al., 2019), maximization of entropy using independent beta distributions (Tiboni et al., 2024), and progressive expansion of a uniform sampling distribution via boundary sampling (OpenAI et al., 2019). Here, we propose a novel learned domain randomization technique using normalizing flows Rezende & Mohamed (2015) as a neural sampling distribution, thus increasing flexibility and expressivity.

In addition to learning robust policies, such sampling distributions can be used as indicators of the world states under which the policy is expected to succeed. Some previous works have combined domain randomization with information gathering via system identification (Ramos et al., 2019; Sagawa & Hino, 2024). In this work, we similarly make use of our learned sampling distribution as an out-of-distribution detector in the context of a multi-step planning system.

3 BACKGROUND

3.1 MARKOV DECISION PROCESS

A Markov Decision Process (MDP) is a mathematical framework for modeling decision-making. Formally, an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, where $P(s' \mid s, a)$ denotes the probability of transitioning to state s' from state s after taking action a, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $R(s, a)$ denotes the expected immediate reward received after taking action a in state s, $\gamma \in [0, 1)$ is the discount factor, representing the importance of future rewards.

A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines a probability distribution over actions given states, where $\pi(a \mid s)$ is the probability of taking action a in state s. The goal is to find an optimal policy π^* that maximizes the expected cumulative discounted reward.

3.2 DOMAIN RANDOMIZATION

Domain randomization introduces variability into the environment by randomizing certain parameters during training. Let Ξ denote the space of domain randomization parameters, and let $\xi \in \Xi$ be a specific instance of these parameters. Each ξ corresponds to a different environment configuration or dynamics.

We can define a parameterized family of Markov Decision Processes (MDPs) where each $\mathcal{M}_\xi = (\mathcal{S}, \mathcal{A}, P_\xi, R_\xi, \gamma)$ has transition dynamics P_ξ and reward function R_ξ dependent on ξ. The agent interacts with environments sampled from a distribution over Ξ, typically denoted as $p(\xi)$. [1]

The objective is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected return across the distribution environments:

$$J(\pi) = \mathbb{E}_{\xi \sim p(\xi)} \left[\mathbb{E}_{\tau \sim P_\xi, \pi} \left[\sum_{t=0}^{\infty} \gamma^t R_\xi(s_t, a_t) \right] \right], \tag{1}$$

where $\tau = \{(s_0, a_0, s_1, a_1, \dots)\}$ denotes a trajectory generated by policy π in environment ξ. Domain randomization aims to find a policy π^* such that: $\pi^* = \arg\max_\pi J(\pi)$.

In deep reinforcement learning, the policy π is a neural network parameterized by θ, denoted as π_θ. The agent learns the policy parameters θ through interactions with simulated environments sampled from $p(\xi)$. In our implementation, we employ the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017), an on-policy policy gradient method that optimizes a stochastic policy while ensuring stable and efficient learning.

To further stabilize training, we pass privileged information about the environment parameters ξ to the critic network. The critic network, parameterized by ψ, estimates the state-value function:

$$V_\psi(s_t, \xi) = \mathbb{E}_{\pi_\theta} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t, \xi \right], \tag{2}$$

where s_t is the current state, r_{t+k} are future rewards, and γ is the discount factor. By incorporating ξ, the critic can provide more accurate value estimates with lower variance (Pinto et al., 2017). The actor network $\pi_\theta(a_t|s_t)$ does not have access to ξ, ensuring that the policy relies only on observable aspects of the state.

3.3 NORMALIZING FLOWS

Normalizing flows are a class of generative models that transform a simple base distribution into a complex target distribution using a sequence of invertible, differentiable functions. Let $z \sim p_Z(z)$ be

[1] This problem can also be thought of as a POMDP where the observation space is $\mathcal{S}$ and the state space is a product of $\mathcal{S}$ and Ξ as discussed in Kwon et al. (2021).

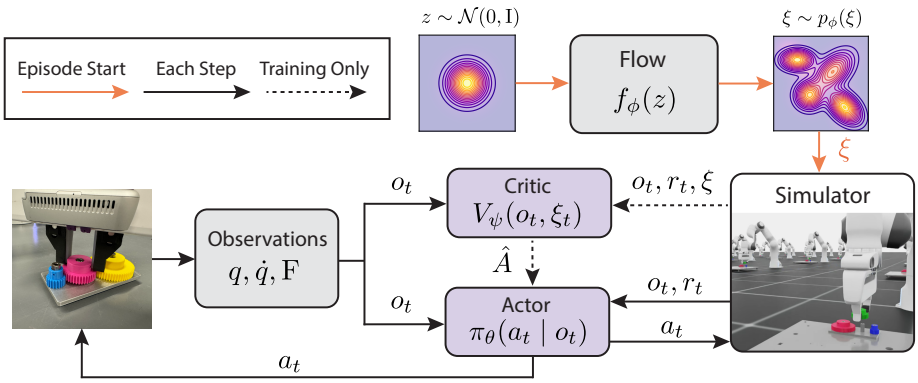

Figure 1: An architecture diagram for our actor-critic RL training setup using a normalizing flow to seed environment parameters across episodes.

a latent variable from a base distribution (e.g., a standard normal distribution). A normalizing flow defines an invertible transformation $f_\phi : \mathbb{R}^d \to \mathbb{R}^d$ parameterized by neural network parameters ϕ, such that $x = f_\phi(z)$, aiming for x to follow the target distribution.

The density of x is computed using the change of variables formula:

$$p_X(x) = p_Z(f_\phi^{-1}(x)) \left| \det \left(\frac{\partial f_\phi^{-1}(x)}{\partial x} \right) \right|. \tag{3}$$

For practical computation, this is often rewritten as:

$$\log p_X(x) = \log p_Z(z) - \log \left| \det \left(\frac{\partial f_\phi(z)}{\partial z} \right) \right|, \tag{4}$$

where $\frac{\partial f_\phi(z)}{\partial z}$ is the Jacobian of f_ϕ at z. By composing multiple such transformations $f_\phi = f_{\phi_K} \circ \cdots \circ f_{\phi_1}$, each parameterized by neural network parameters ϕ_k, normalizing flows can model highly complex distributions.

In our work, we employ *neural spline flows* (Durkan et al., 2019), a type of normalizing flow where the invertible transformations are constructed using spline-based functions. Specifically, the parameters ϕ represent the coefficients of the splines (e.g., knot positions and heights) and the weights and biases of the neural networks that parameterize these splines.

4 Method

In this section, we introduce **GoFlow**, a method for learned domain randomization that *goes with the flow* by adaptively adjusting the domain randomization process using normalizing flows.

In traditional domain randomization setups, the distribution $p(\xi)$ is predefined. However, selecting an appropriate $p(\xi)$ is crucial for the policy's performance and generalization. Too broad a sampling distribution and the training focuses on unsolvable environments and falls into local minima. In contrast, too narrow a sampling distribution leads to poor generalization and robustness. Additionally, rapid changes to the sampling distribution can lead to unstable training. To address these challenges, prior works such as Klink et al. (2021) have proposed a self-paced learner, which starts by mastering a small set of environments, and gradually expands the tasks to solver harder and harder problems while maintaining training stability. This strategy has subsequently been applied to domain randomization in Mozifian et al. (2019) and Tiboni et al. (2024), where terms were included for encouraging spread over the sampling space for greater generalization. We take inspiration from these works to form a joint optimization problem:

Algorithm 1 GoFlow

Require: Initial policy parameters θ, flow parameters ϕ, total training steps N, network updates K, Monte Carlo samples M, entropy coefficient α, similarity coefficient β, and learning rate η_ϕ

1: **for** $n = 1$ to N **do**
2: Sample $\{\xi_i\}_{i=1}^{B} \sim p_\phi(\xi)$
3: Train π_θ with ξ_i initializations
4: Estimate $J_{\xi_i}(\pi_\theta)$ for each ξ_i via M Monte Carlo samples
5: Save current flow distribution as $p_{\phi_{\text{old}}}(\xi)$
6: **for** $k = 1$ to K **do**
7: $\hat{\mathcal{H}} \leftarrow -\mathbb{E}_{\xi \sim p_\phi(\xi)} \left[\log p_\phi(\xi)\right]$
8: $\hat{D}_{KL} \leftarrow \mathbb{E}_{\xi \sim p_\phi(\xi)} \left[\log p_\phi(\xi) - \log p_{\phi_{\text{old}}}(\xi)\right]$
9: $\phi \leftarrow \phi + \eta_\phi \nabla_\phi \left(\frac{1}{B} \sum_i^B \left[\log p_\phi(\xi) J_{\xi_i}(\pi_\theta)\right] + \alpha \hat{H} - \beta \hat{D}_{KL} \right)$
10: **end for**
11: **end for**

$$\max_{p(\xi),\pi} \left\{ \mathbb{E}_{\xi \sim p(\xi)} \left[J_\xi(\pi) \right] + \alpha \mathcal{H}(p(\xi)) - \beta D_{KL} \left(p(\xi) \| p_{\text{old}}(\xi) \right) \right\}, \tag{5}$$

where $\mathcal{H}(p(\xi))$ is the differential entropy of $p(\xi)$, $D_{KL}\left(p(\xi)\|p_{\text{old}}(\xi)\right)$ is the divergence between the current and previous sampling distributions, and $\alpha > 0, \beta > 0$ are regularization coefficients that control the tradeoff between generalizability, training stability, and the expected reward under the sampling distribution. To our knowledge, GoFlow is the first method to optimize such an objective with a neural sampling distribution.

Other learned domain randomization approaches propose similar objectives. Mozifian et al. (2019) maximizes reward but replaces entropy regularization with a KL divergence to a fixed target distribution and omits the self-paced KL term. Tiboni et al. (2024) includes all three objectives but frames the reward and self-paced KL terms as constraints, maximizing entropy through a nonlinear optimization process that is not easily adaptable to neural sampling distributions. We compare GoFlow to these methods in our experiments to highlight its advantages.

The GoFlow algorithm (Algorithm 1) begins by initializing both the policy parameters θ and the normalizing flow parameters ϕ. In each training iteration, GoFlow first samples a batch of environment parameters $\{\xi_i\}_{i=1}^{B}$ from the current distribution modeled by the normalizing flow. These sampled parameters are used to train the policy π_θ. Following the policy update, expected returns $J_{\xi_i}(\pi_\theta)$ are estimated for each sampled environment through Monte Carlo sampling, providing a measure of the policy's performance in each specific setting.

After sampling these rollouts, GoFlow then perform K steps of optimization on the sampling distribution. The entropy of the sampling distribution (Line 7) and divergence from the previous sampling distribution (Line 8) are estimated using newly drawn samples. GoFlow then estimates the policy's performance on the sampling distribution using the previously sampled rollout trajectories. Since we are evaluating the gradient, this can equivalently be calculated using the log of the probability under the sampling distribution for numerical stability, as is commonly done in policy gradient reinforcement learning (see proof in Appendix Section A.7). These terms are combined to form a loss, which is differentiated to update the parameters of the sampling distribution (Line 9). An architecture diagram for this approach can be seen in Figure 14.

5 Domain Randomization Experiments

Our simulated experiments compare policy robustness in a range of domains. For full details on the randomization parameters and bounds, see Appendix A.1.

5.1 Domains

First, we examine the application of GoFlow to an illustrative 2D domain that is multimodal and contains intervariable dependencies. The state and action space are in $\mathbb{R}^2$. The agent is initialized

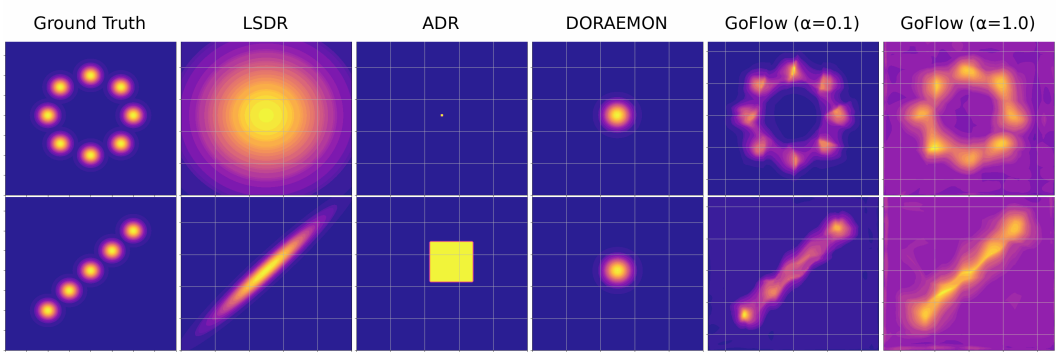

Figure 2: An illustrative domain showing the learned sampling functions over the space of unobserved parameters for the tested baselines. Compared to other learning methods, GoFlow correctly models the multimodality and inter-variable dependencies of the underlying reward function. This toy domain, along with other domains in our experiments, violates some of the assumptions made by prior works, such as the feasibility of the center point of the range.

randomly in a bounded x, y plane. An energy function is defined by a composition of Gaussians placed in a regular circular or linear array. The agent can observe its position with Gaussian noise proportional to the inverse of the energy function. The agent is rewarded for guessing its location, but is incapable of moving. This task is infeasible when the agent is sufficiently far from any of the Gaussian centers, so a sampling distribution should come to resemble the energy function. Some example functions learned by GoFlow and baselines from Section 5.2 can be seen in Figure 2.

Second, we quantitatively compare GoFlow to existing baselines including Cartpole, Ant, Quadcopter, and Quadruped in the IsaacLab suite of environments (Mittal et al., 2023). We randomize over parameters such as link masses, joint frictions, and material properties.

Lastly, we evaluate our method on a contact-rich robot manipulation task of gear insertion, a particularly relevant problem for robotic assembly. In the gears domain, we randomize over the relative pose between the gripper and the held gears along three degrees of freedom. The problem is made difficult by the uncertainty the robot has about the precise location of the gear relative to the hand. The agent must learn to rely on signals of proprioception and force feedback to guide the gear into the gear shaft. The action space consists of end-effector pose offsets along three translational degrees of freedom and one rotational degree around the z dimension. The observation space consists of a history of the ten previous end effector poses and velocities estimated via finite differencing. In addition to simulated experiments, our trained policies are tested on a Franka Emika robot using `IndustReal` library built on the `frankapy` toolkit (Tang et al., 2023b; Zhang et al., 2020).

5.2 Baselines

In our domain randomization experiments, we compare to a number of standard RL baselines and learning-based approaches from the literature. In our quantitative experiments, success is measured by the sampled environment passing a certain performance threshold J_T that was selected for each environment. All baselines are trained with an identical neural architecture and PPO implementation. The success thresholds along with other hyperparameters are in Appendix A.2.

We evaluate on the following baselines. First, we compare to no domain randomization (**NoDR**) which trains on a fixed environment parameter at the centroid of the parameter space. Next, we compare to a full domain randomization (**FullDR**) which samples uniformly across the domain within the boundaries during training. In addition to these fixed randomization methods, we evaluate against some other learning-based solutions from the literature: **ADR** (OpenAI et al., 2019) learns uniform intervals that expand over time via boundary sampling. It starts by occupying an initial percentage of the domain and performs "boundary sampling" during training with some probability. The rewards attained from boundary sampling are compared to thresholds that determine if the boundary should be expanded or contracted. **LSDR** (Mozifian et al., 2019) learns a multivariate

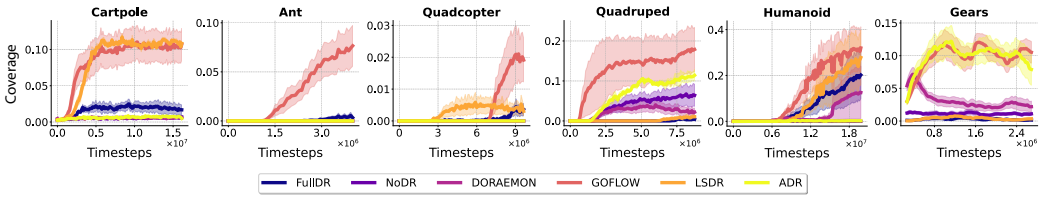

Figure 3: The coverage ratio over the target distribution across five random seeds for each of the environments. The bands around each curve indicate the standard error.

gaussian sampling distribution using reward maximization with a KL divergence regularization term weighted by an α hyperparameter. Lastly, **DORAEMON** (Tiboni et al., 2024) learns independent beta distributions for each dimension of the domain, using a maximum entropy objective constrained by an estimated success rate.

We measure task performance in terms of coverage, which is defined as the proportion of the total sampling distribution for which the policy receives higher than J_t reward. We compare coverage across environments sampled from a uniform testing distribution within the environment bounds. Our findings in Figure 3 show that GoFlow matches or outperforms baselines across all domains. We find that our method performs particularly well in comparison to other learned baselines when the simpler or more feasible regions of the domain are off-center, irregularly shaped, and have inter-parameter dependencies such as those seen in Figure 2. The reward structure and sampling ranges of these domains often violate the regularity assumptions made by the baseline approaches, leading to worse performance.

In addition to simulated experiments, we additionally evaluated the trained policies on a real-world gear insertion task. The results of those real-world experiments can be seen in Table 2.

6 Application to Multi-step manipulation

While reinforcement learning has proven to be a valuable technique for learning short-horizon skills in dynamic and contact-rich settings, it often struggles to generalize to more long-horizon and open ended problems (Sutton & Barto, 2018). The topic of sequencing short horizon skills in the context of a higher-level decision strategy has been of increasing interest to both the planning (Mishra et al., 2023) and reinforcement learning communities (Nasiriany et al., 2021). For this reason, we examine the utility of these learned sampling distributions as out-of-distribution detectors, or belief-space preconditions, in the context of a multi-step planning system.

6.1 Belief-space planning background

Belief-space planning is a framework for decision-making under uncertainty, where the agent maintains a probability distribution over possible states, known as the *belief state*. Instead of planning solely in the state space $\mathcal{S}$, the agent operates in the *belief space* $\mathcal{B}$, which consists of all possible probability distributions over $\mathcal{S}$. This approach is particularly useful in partially observable environments where there is uncertainty in environment parameters and where it is important to take actions to gain information.

Rather than operating at the primitive action level, belief-space planners often make use of high-level actions $\mathcal{A}_\Pi$, sometimes called skills or options. In our case, these high-level actions will be a discrete set of pretrained RL policies. These high-level actions come with a *belief-space precondition* and a *belief-space effect*, both of which are subsets of the belief space $\mathcal{B}$ (Kaelbling & Lozano-Perez (2013); Curtis et al. (2024)). Specifically, a high-level action $\pi \in \mathcal{A}_\Pi$ is associated with two components: a precondition $\text{Pre}_\pi \subseteq \mathcal{B}$, representing the set of belief states from which the action can be applied, and an effect $\text{Eff}_\pi \subseteq \mathcal{B}$, representing the set of belief states that the action was designed or trained to achieve. If the precondition holds—that is, if the current belief state b satisfies $b \in \text{Pre}_\pi$—then applying action a will achieve the effect Eff_π with probability at least $\eta \in [0, 1]$.

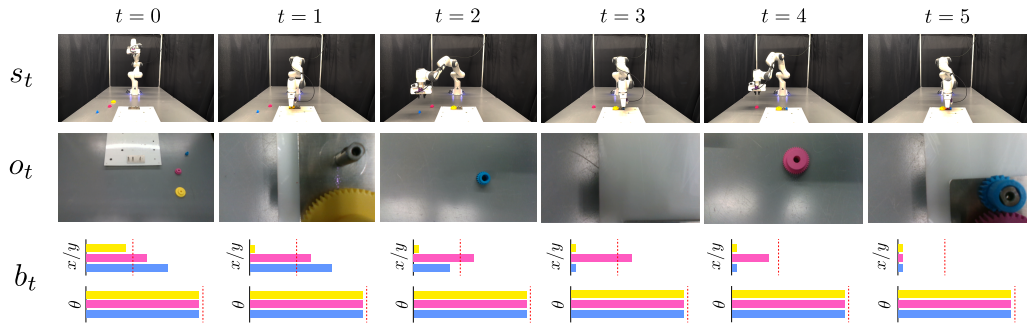

Figure 4: A multi-step manipulation plan using probabilistic pose estimation to estimate and update beliefs over time. The three rows show the robot state s_t, the observation o_t, and the robot belief b_t at each timestep. The red dotted line in the belief indicates the marginal entropy thresholds for the x, y, and yaw dimensions as determined by the learned normalizing flow. For full visualizations of the belief posteriors, flow distributions, and value maps, see Figure 12 in the appendix.

More formally, if $b \in \text{Pre}_\pi$, then $\Pr\left(b_{t+1} \in \text{Eff}_\pi \,\middle|\, b_t, \pi\right) \geq \eta$. Depending on the planner, η may be set in advance, or calculated by the planner as a function of the belief.

This formalization allows planners to reason abstractly about the effects of high-level actions under uncertainty, which can result in generalizable decision-making in long-horizon problems that require active information gathering or risk-awareness.

6.2 Computing Preconditions

In this section, we highlight the potential application of the learned sampling distribution p_ϕ and privileged value function V_ψ as a useful artifacts for belief-space planning. In particular, we are interested in identifying belief-space preconditions of a set of trained skills.

One point of leverage we have for this problem is the privileged value function $V_\psi(s, \xi)$, which was learned alongside the policy during training. One way to estimate the belief-space precondition is to simply find the set of belief states for which the expected value of the policy is larger than J_T with probability greater than η under the belief:

$$\text{Pre}_\pi = \left\{b(s,\xi) \in \mathcal{B} \mid \mathbb{E}_{b(s,\xi)}\left[\mathbb{1}_{V_\psi(s,\xi) > J_T}\right] > \eta\right\}. \tag{6}$$

However, a practical issue with this computation is that the value function is likely not calibrated in large portions of the state space that were not seen during policy training. To address this, we focus on regions of the environment where the agent has higher confidence due to sufficient sampling, i.e., where $p_\phi(\xi) > \epsilon$ for a threshold ϵ. This enables us to integrate the value function over the belief distribution $b(x, \xi)$ and the trusted region within Ξ:

$$\text{Pre}_\pi = \left\{b(s,\xi) \in \mathcal{B} \mid \mathbb{E}_{b(s,\xi)}\left[\mathbb{1}_{V_\psi(s,\xi) > J_T} \cdot \mathbb{1}_{p_\phi(\xi) > \epsilon}\right] > \eta\right\} \tag{7}$$

.

Here, η lower bounds the probability of achieving the desired effects Eff_π (or value greater than J_T) after executing π in any belief state in Pre_π. Figure 5 shows an example precondition for a single step of the assembly plan.

6.3 Updating Beliefs

Updating the belief state requires a probabilistic state estimation system that outputs a posterior over the unobserved environment variables, rather than a single point estimate. We use a probabilistic object pose estimation framework called Bayes3D to infer posterior distributions over object pose (Gothoskar et al., 2023). For details on this, see Appendix A.4.1.

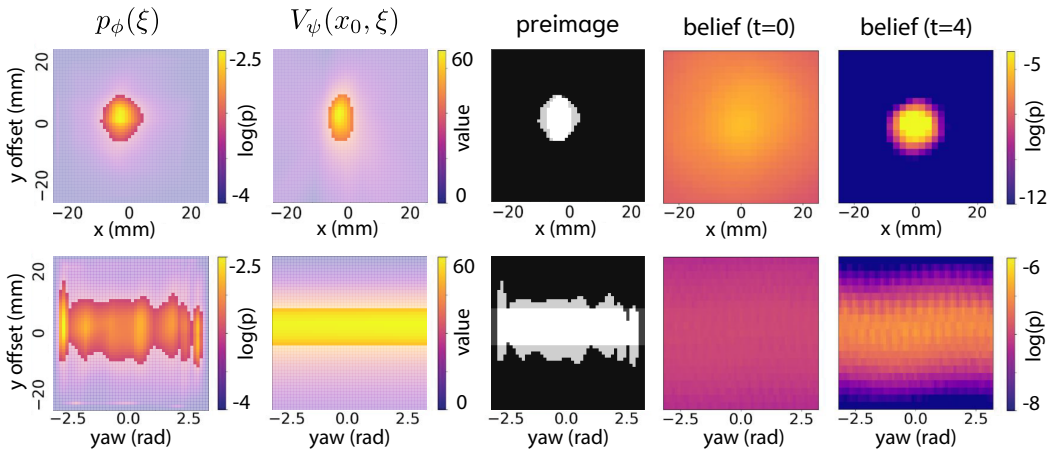

Figure 5: A visual example of the precondition computation described in Section 6.2 for a single step in the gear assembly process. The normalizing flow is thresholded to indicate the in-distribution regions of the learned value function. The thresholded sampling distribution is further thresholded by the value function to get the belief-space precondition. Comparing the precondition to the beliefs, we can see that the belief is not sufficiently contained within the precondition at $t = 0$, but passes the success threshold η at after closer inspection $t = 4$.

The benefit of this approach in contrast to traditional rendering-based pose estimation systems, such as those presented in Wen et al. (2024) or Labbé et al. (2022), is that pose estimates from Bayes3D indicate high uncertainty for distant, small, or occluded objects as well as uncertainty stemming from object symmetry. Figure 12 shows the pose beliefs across the multi-step plan.

6.4 A SIMPLE BELIEF-SPACE PLANNER

While the problem of general-purpose multi-step planning in belief-space has been widely studied, in this paper we use a simple BFS belief-space planner to demonstrate the utility of the learned sampling distributions as belief-space preconditions. The full algorithm can be found in Algorithm 2.

An example plan can be seen in Figure 4. The goal is to assemble the gear box by inserting all three gears (yellow, pink, and blue) into the shafts on the gear plate. Each gear insertion is associated with a separate policy for each color trained with GoFlow. In addition to the trained policies, the robot is given access to an object-parameterized inspection action which has no preconditions and whose effects are a reduced-variance pose estimate attained by moving the camera closer to the object. The robot is initially uncertain of the x, y, and yaw components of the 6-dof pose based on probabilistic pose estimates. Despite this uncertainty, the robot is confident enough in the pose of the largest and closest yellow gear to pick it up and insert it. In contrast, the blue and pink gears require further inspection to get a better pose estimate. Closer inspection reduces uncertainty along the x and y axis, but reveals no additional information about yaw dimension due to rotational symmetry. Despite an unknown yaw dimension, the robot is confident in the insertion because the flow p_ϕ indicates that success is invariant to the yaw dimension. For visualizations of the beliefs and flows at each step, see Appendix A.4.

7 CONCLUSION AND DISCUSSION

In this paper, we introduced GoFlow, a novel approach to domain randomization that uses normalizing flows to dynamically adjust the sampling distribution during reinforcement learning. By combining actor-critic reinforcement learning with a learned neural sampling distribution, we enabled more flexible and expressive parameterization of environmental variables, leading to better generalization in complex tasks like contact-rich assembly. Our experiments demonstrated that GoFlow outperforms traditional fixed and learning-based domain randomization techniques across a variety

of simulated environments, particularly in scenarios where the domain has irregular dependencies between parameters. The method also showed promise in real-world robotic tasks including contact-rich assembly.

Moreover, we extended GoFlow to multi-step decision-making tasks, integrating it with belief-space planning to handle long-horizon problems under uncertainty. This extension enabled the use of learned sampling distributions and value functions as preconditions leading to active information gathering.

Although GoFlow enables more expressive sampling distributions, it also presents some new challenges. One limitation of our method is that it has higher variance due to occasional training instability of the flow. This instability can be alleviated by increasing β, but at the cost of reduced sample efficiency (see Appendix A.2). In addition, using the flow and value estimates for belief-space planning require manual selection and tuning of several thresholds which are environment specific. The η parameter may be converted from a threshold into a cost in the belief space planner, which would remove one point of manual tuning. However, removing the ϵ parameter may prove more difficult, as it would require uncertainty quantification of the neural value function. Despite these challenges, we hope this work inspires further research on integrating short-horizon learned policies into broader planning frameworks, particularly in contexts involving uncertainty and partial observability.

8 Reproducibility Statement

In our supplementary materials, we provide a minimal code implementation of our approach along with all the baseline implementations and environments discussed in this paper. If accepted, we will publish a full public release of the code on Github.

References

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979. IEEE, 2019.

Xiaoyu Chen, Jiachen Hu, Chi Jin, Lihong Li, and Liwei Wang. Understanding domain randomization for sim-to-real transfer. *CoRR*, abs/2110.03239, 2021. URL https://arxiv.org/abs/2110.03239.

Aidan Curtis, George Matheos, Nishad Gothoskar, Vikash Mansinghka, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Partially observable task and motion planning with uncertainty and risk awareness, 2024. URL https://arxiv.org/abs/2403.10454.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, 2006.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Nishad Gothoskar, Matin Ghavami, Eric Li, Aidan Curtis, Michael Noseworthy, Karen Chung, Brian Patton, William T Freeman, Joshua B Tenenbaum, Mirko Klukas, et al. Bayes3d: fast learning and inference in structured generative models of 3d objects and scenes. *arXiv preprint arXiv:2312.08715*, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Peng Jin, Yu Lin, Yu Song, Tao Li, and Wei Yang. Vision-force-fused curriculum learning for robotic contact-rich assembly tasks. *Frontiers in Neurorobotics*, 17:1280773, October 2023. doi: 10.3389/fnbot.2023.1280773.

Josip Josifovski, Mohammadhossein Malmir, Noah Klarmann, Bare Luka Žagar, Nicolás Navarro-Guerrero, and Alois Knoll. Analysis of randomization effects on sim2real transfer in reinforcement learning for robotic manipulation tasks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10193–10200. IEEE, 2022.

Leslie Kaelbling and Tomas Lozano-Perez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32:1194–1227, 08 2013. doi: 10.1177/0278364913484072.

Pascal Klink, Hany Abdulsamad, Boris Belousov, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. A probabilistic interpretation of self-paced learning with applications to reinforcement learning. *CoRR*, abs/2102.13176, 2021. URL https://arxiv.org/abs/2102.13176.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Jeongyeol Kwon, Yonathan Efroni, Constantine Caramanis, and Shie Mannor. RL for latent mdps: Regret guarantees and a lower bound. *CoRR*, abs/2102.04939, 2021. URL https://arxiv.org/abs/2102.04939.

Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. Megapose: 6d pose estimation of novel objects via render & compare, 2022. URL https://arxiv.org/abs/2212.06870.

Jieliang Luo and Hui Li. A learning approach to robot-agnostic force-guided high precision assembly. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2151–2157. IEEE, 2021.

Utkarsh A. Mishra, Shangjie Xue, Yongxin Chen, and Danfei Xu. Generative skill chaining: Long-horizon skill planning with diffusion models, 2023. URL https://arxiv.org/abs/2401.03360.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Melissa Mozifian, Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. Learning domain randomization distributions for transfer of locomotion policies. *CoRR*, abs/1906.00410, 2019. URL http://arxiv.org/abs/1906.00410.

Fabio Muratore, Michael Gienger, and Jan Peters. Assessing transferability from simulation to reality for reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 43(4):1172–1183, 2019.

Fabio Muratore, Theo Gruner, Florian Wiese, Boris Belousov, Michael Gienger, and Jan Peters. Neural posterior domain randomization. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1532–1542. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/muratore22a.html.

Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks. *CoRR*, abs/2110.03655, 2021. URL https://arxiv.org/abs/2110.03655.

Michael Noseworthy, Bingjie Tang, Bowen Wen, Ankur Handa, Nicholas Roy, Dieter Fox, Fabio Ramos, Yashraj Narang, and Iretiayo Akinola. Forge: Force-guided exploration for robust contact-rich manipulation under uncertainty, 2024. URL https://arxiv.org/abs/2408.04587.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018. URL http://arxiv.org/abs/1810.12282.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *CoRR*, abs/1710.06542, 2017. URL http://arxiv.org/abs/1710.06542.

Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *CoRR*, abs/1906.01728, 2019. URL http://arxiv.org/abs/1906.01728.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. PMLR, 2015.

François Rozet et al. Zuko: Normalizing flows in pytorch, 2022. URL https://pypi.org/project/zuko.

Shogo Sagawa and Hideitsu Hino. Gradual domain adaptation via normalizing flows, 2024. URL https://arxiv.org/abs/2206.11492.

Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5548–5555. IEEE, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Bingjie Tang, Michael A. Lin, Iretiayo Akinola, Ankur Handa, Gaurav S. Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality, 2023a. URL https://arxiv.org/abs/2305.17110.

Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality. In *Robotics: Science and Systems*, 2023b.

Gabriele Tiboni, Karol Arndt, and Ville Kyrki. Dropo: Sim-to-real transfer with offline domain randomization. *Robotics and Autonomous Systems*, pp. 104432, 2023. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2023.104432. URL https://www.sciencedirect.com/science/article/pii/S0921889023000714.

Gabriele Tiboni, Pascal Klink, Jan Peters, Tatiana Tommasi, Carlo D'Eramo, and Georgia Chalvatzaki. Domain randomization via entropy maximization, 2024. URL https://arxiv.org/abs/2311.01885.

Eugene Valassakis, Zihan Ding, and Edward Johns. Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5372–5379. IEEE, 2020.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects, 2024. URL https://arxiv.org/abs/2312.08344.

Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020.

Xiang Zhang, Masayoshi Tomizuka, and Hui Li. Bridging the sim-to-real gap with dynamic compliance tuning for industrial insertion, 2024. URL https://arxiv.org/abs/2311.07499.

Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

A APPENDIX

A.1 DOMAIN RANDOMIZATION PARAMETERS

Below we describe the randomization ranges and parameter names for each environment. We also provide the reward success threshold (J_T) and cut the max duration of some environments in order to speed up training (t_{max}). Lastly, we slightly modified the Quadruped environment to only take a fixed forward command rather than the goal-conditioned policy learned by default. Other than those changes, the first four simulated environments official IsaacLab implementation.

- **Cartpole randomization parameters ($J_T = 50, t_{max} = 2s$):**
  - Pole mass: Min Bound = 0.01, Max Bound = 20.0
  - Cart mass: Min Bound = 0.01, Max Bound = 20.0
  - Slider-Cart Friction: Min Bound = 0.0, Max Bound = 1.0
- **Ant randomization parameters ($J_T = 700, t_{max} = 2s$):**
  - Torso mass: Min Bound = 0.01, Max Bound = 20.0
- **Quadcopter randomization parameters ($J_T = 15, t_{max} = 2s$):**
  - Quadcopter mass: Min Bound = 0.01, Max Bound = 20.0
- **Quadruped randomization parameters ($J_T = 1.5, t_{max} = 5s$):**
  - Body mass: Min Bound = 0.0, Max Bound = 200.0
  - Left front hip joint friction: Min Bound = 0.0, Max Bound = 0.1
  - Left back hip joint friction: Min Bound = 0.0, Max Bound = 0.1
  - Right front hip joint friction: Min Bound = 0.0, Max Bound = 0.1
  - Right back hip joint friction: Min Bound = 0.0, Max Bound = 0.1
- **Humanoid randomization parameters ($J_T = 1000, t_{max} = 5s$):**
  - Torso Mass: Min Bound = 0.01, Max Bound = 25.0
  - Head Mass: Min Bound = 0.01, Max Bound = 25.0
  - Left Hand Mass: Min Bound = 0.01, Max Bound = 30.0
  - Right Hand Mass: Min Bound = 0.01, Max Bound = 30.0
- **Gear randomization parameters ($J_T = 50, t_{max} = 4s$):**
  - Grasp Pose x: Min Bound = -0.05, Max Bound = 0.05
  - Grasp Pose y: Min Bound = -0.05, Max Bound = 0.05
  - Grasp Pose yaw: Min Bound = -0.393, Max Bound = 0.393

A.2 HYPERPARAMETERS

Below we list out the significant hyperparameters involved in each baseline method, and how we chose them based on our hyperparameter search. We run the same seed for each hyperparameter and pick the best performing hyperparameter as the representative for our larger quantitative experiments in figure 3. The full domain randomization (FullDR) and no domain randomization (NoDR) baselines have no hyperparameters.

A.2.1 GOFLOW

We search over the following values of the α hyperparameter: $[0.1, 0.5, 1.0, 1.5, 2.0]$. We search over the following values. Other hyperparameters include number of network updates per training epoch ($K = 100$), network learning rate ($\eta_\phi = 1e{-}3$), and neural spline flow architecture hyperparameters such as network depth ($\ell = 3$), hidden features (64), and number of bins (8). We implement our flow using the Zuko normalizing flow library Rozet et al. (2022).

A.2.2 LSDR

Similary to GoFlow , we search over the following values of the α_L hyperparameter: $[0.1, 0.5, 1.0, 1.5, 2.0]$. Other hyperparameters include the number of updates per training epoch (T=100), and initial Gaussian parameters: $\mu = (\xi_{max} + \xi_{min})/2.0$ and $\Sigma = \text{diag}\,(\xi_{\text{max}} - \xi_{\text{min}}/10)$

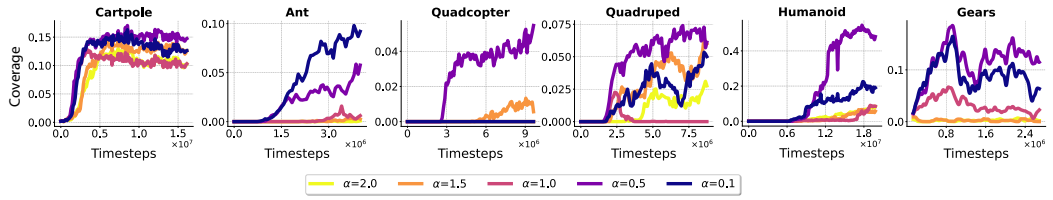

Figure 6: GoFlow hyperparameter sweep results for α

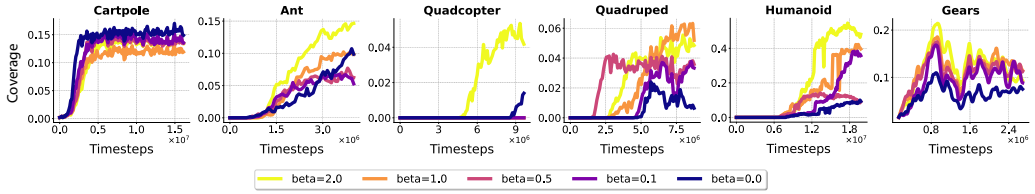

Figure 7: GoFlow hyperparameter sweep results for β after fixing the best α for each environment

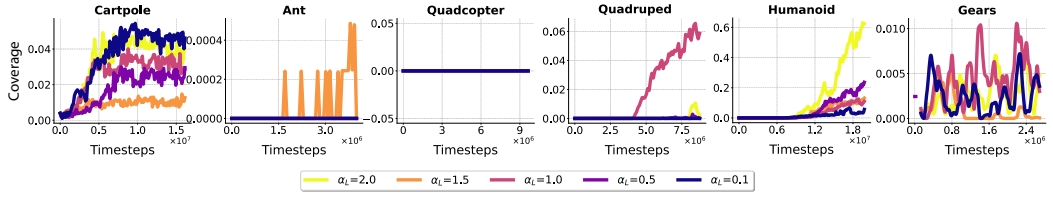

Figure 8: LSDR hyperparameter sweep results for α_L

A.2.3 DORAEMON

We search over the following values of the ϵ_D hyperparameter: $[0.005, 0.01, 0.05, 0.1, 0.5]$. After fixing the best ϵ_D for each environment, we additionally search over the success thrshold α_D: $[0.1, 0.25, 0.5, 0.75, 0.9]$.

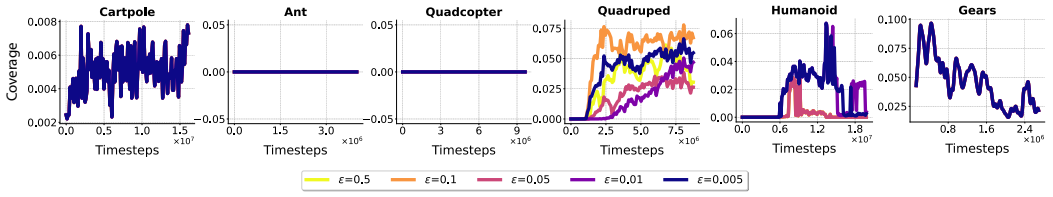

Figure 9: DORAEMON hyperparameter sweep results for ϵ_D

A.2.4 ADR

In ADR, we fix the upper threshold to be the success threshold $t^+ = J_T$ as was done in the original paper and search over the lower bound threshold $t^- = [0.1t^+, 0.25t^+, 0.5t^+, 0.75t^+, 0.9t^+]$. The value used in the original paper was $0.5t_H$. Other hyperparameters include the expansion/contraction rate, which we interpret to be a fixed fraction of the domain interval, $\Delta = 0.1 * \left[\xi_{\text{max}} - \xi_{\text{min}}\right]$, and boundary sampling probability $p_b = 0.5$.

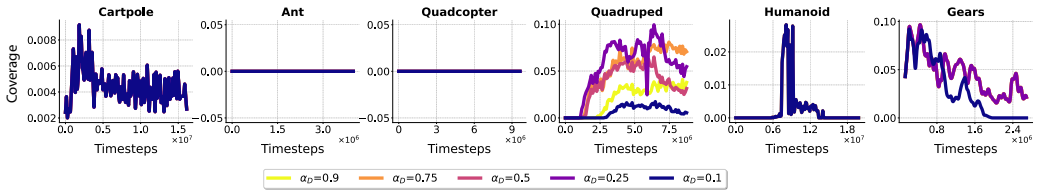

Figure 10: DORAEMON hyperparameter sweep results for α_D after fixing the best ϵ_D

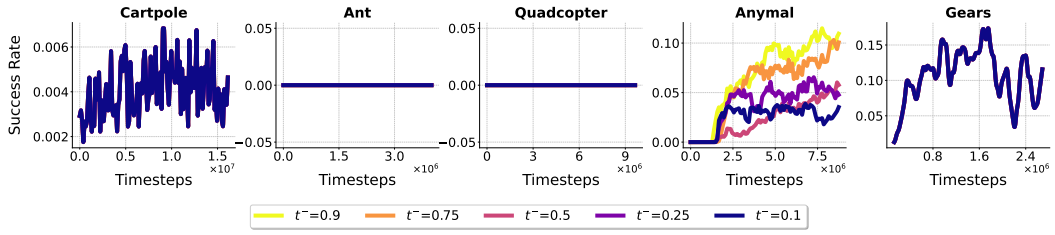

Figure 11: ADR hyperparameter sweep results for t^-

A.3 Coverage vs Range Experiments

We compare coverage vs. range scale in the ant domain. We adjust the parameter lower and upper bounds outlined in Appendix A.1 and see how the coverage responds to those changes during training. The parameter range is defined relative to a nominal midpoint m set to the original domain parameters: $[m - (m - \text{lower}) * \text{scale}, m + (\text{upper} - m) * \text{scale})]$. The results of our experiment are shown in Figure 13

A.4 Multi-Step Planning Details

A.4.1 Updating beliefs via probabilistic pose estimation

Updating the belief state b requires a probabilistic state estimation system that outputs a posterior over the state space S, rather than a single point estimate. Given a new observation o, we use a probabilistic object pose estimation framework (Bayes3D) to infer posterior distributions over object pose (Gothoskar et al., 2023).

The pose estimation system uses inference in an probabilistic generative graphics model with uniform priors on the translational x, y, and rotational yaw (or r_x) components of the 6-dof pose (since the object is assumed to be in flush contact with the table surface) and an image likelihood $P(o_{\text{rgbd}} \mid r_x, x, y)$. The object's geometry and color information is given by a mesh model. The image likelihood is computed by rendering a latent image im_{rgbd} with the object pose corresponding to (r_x, x, y) and calculating the per-pixel likelihood:

$$P(o_{\text{rgbd}} \mid r_x, x, y) \propto \prod_{i,j \in C} \left[p_{\text{out}} + (1 - p_{\text{out}}) \cdot P_{\text{in}}(o_{\text{rgbd}}[i,j] \mid r_x, x, y)\right] \tag{8}$$

$$P_{\text{in}}(o_{\text{rgbd}}[i,j] \mid r_x, x, y) \propto \exp\left(-\frac{||o_{\text{rgb}}[i,j] - im_{\text{rgb}}[i,j]||_1}{b_{\text{rgb}}} - \frac{||o_{\text{d}}[i,j] - im_{\text{d}}[i,j]||_1}{b_{\text{d}}}\right) \tag{9}$$

where i and j are pixel row and column indices, C is the set of valid pixels returned by the renderer, b_{rgb} and b_{d} are hyperparameters that control posterior sensitivity to the color and depth channels, and p_{out} is the pixel outlier probability hyperparameter. For an observation o_{rgbd}, we can sample from $P(r_x, x, y \mid o_{\text{rgbd}}) \propto P(o_{\text{rgbd}} \mid r_x, x, y)$ to recover the object pose posterior with a tempering

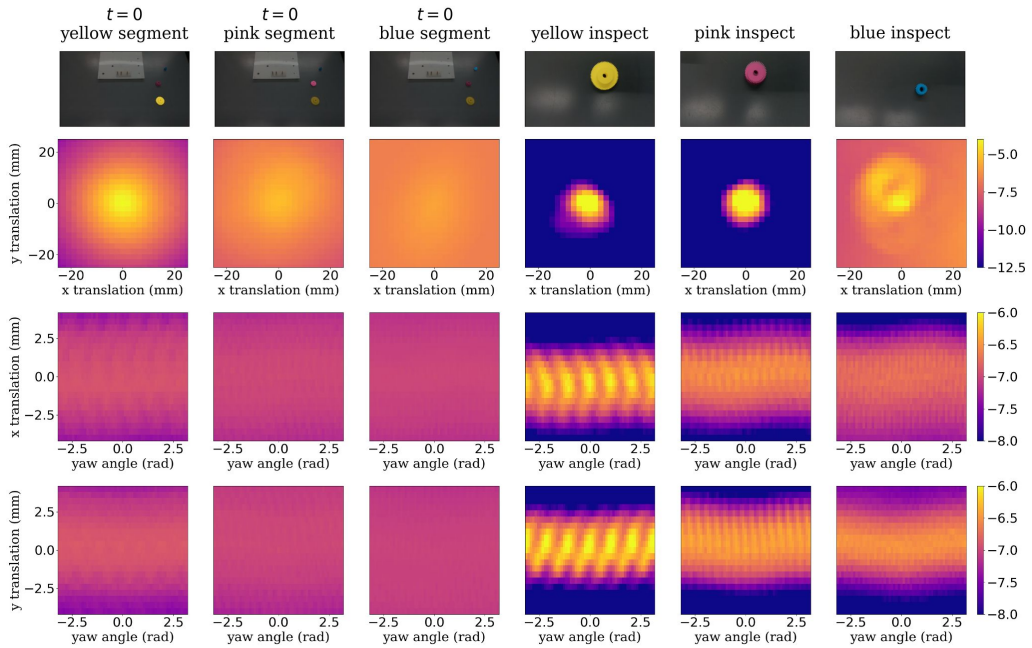

Figure 12: A visualization of the beliefs over the object pose under the initial image (first three columns) and after closer inspection (last three columns) as generated from the posterior of the model described in Section 6.3. The colormap corresponds to the log probability of the posterior pose estimate. All plots are centered around the most likely pose estimate under the image model.

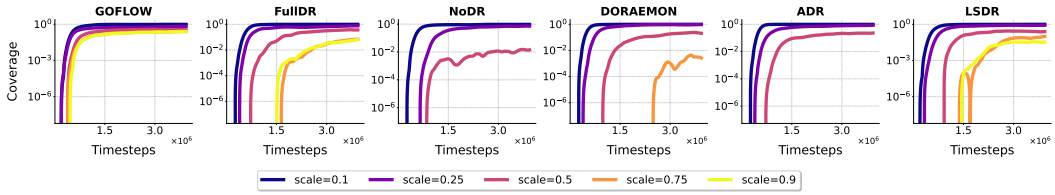

Figure 13: Coverage vs range experiment results

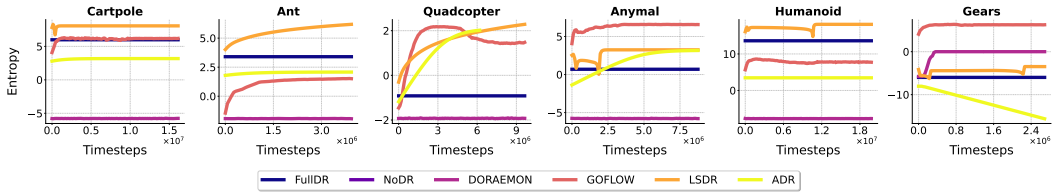

Figure 14: Entropy during the training process for each method

exponential factor α to encourage smoothness. We first find the maximum a posteriori (MAP) estimate of object pose using coarse-to-fine sequential Monte Carlo sampling (Del Moral et al., 2006) and then calculate a posterior approximation using a grid centered at the MAP estimate.

The benefit of this approach in contrast to traditional rendering-based pose estimation systems, such as those presented in Wen et al. (2024) or Labbé et al. (2022), is that our pose estimates indicate

high uncertainty for distant, small, occluded, or non-visible objects as well dimensions along which the object is symmetric. A visualization of the pose beliefs at different points in the multi-step plan can be seen in Figure 12 in the Appendix.

A.4.2 BELIEF-SPACE PLANNING ALGORITHM

Algorithm 2 Belief-Space Planner Using BFS

Require: Initial belief state b_0, goal condition $G \subseteq \mathcal{B}$, set of skills $\mathcal{A}_\Pi$, success threshold η

Initialize the frontier $\mathcal{F} \leftarrow \{b_0\}$
Initialize the visited set $\mathcal{V} \leftarrow \emptyset$
Initialize the plan dictionary Plan mapping belief states to sequences of skills
while $\mathcal{F}$ is not empty **do**
    Dequeue b from $\mathcal{F}$
    if $b \in G$ **then**
        return Plan$[b]$ ▷ Return the sequence of skills leading to b
    end if
    for all skills $\pi \in \mathcal{A}_\Pi$ **do**
        if $b \in \text{Pre}_\pi$ given η **then**
            $b' \leftarrow \texttt{sample}(\text{Eff}_\pi)$
            if $b' \notin \mathcal{V}$ **then**
                Add b' to $\mathcal{F}$ and $\mathcal{V}$
                Update Plan$[b'] \leftarrow$ Plan$[b] + [\pi]$
            end if
        end if
    end for
end while
return Failure ▷ No plan found

	Cartpole	Ant	Quadcopter	Quadruped	Humanoid	Gears
FullDR	.017±.012	.003±.002	.003±.004	.001±.000	.200±.198	.001±.000
NoDR	.005±.000	.000±.000	.000±.000	**.062±.048**	.000±.000	.010±.000
DORAEMON	.005±.000	.000±.000	.000±.000	**.023±.022**	**.124±.248**	.020±.018
LSDR	**.110±.006**	.000±.000	.002±.002	.011±.002	**.278±.230**	.003±.000
ADR	.004±.000	.000±.000	.000±.000	**.114±.026**	.000±.000	**.072±.044**
GoFlow	**.109±.080**	**.072±.038**	**.019±.010**	**.178±.220**	**.321 ± .068**	**.104±.028**

Table 1: Mean and standard deviation of coverage, with all statistically significant entries bolded.

A.5 REAL-WORLD EXPERIMENTS

In addition to simulated experiments, we compare GoFlow against baselines on a real-world gear insertion task. In particular, we tested insertion of the pink medium gear over 10 trials for each baseline. To test this, we had the robot perform 10 successive pick/inserts of the pink gear into the middle shaft of the gear plate. Instead of randomizing the pose of the gear, we elected to fix the initial pose of the gear and the systematically perturb the end-effector pose by a random ± 0.01 meter translational offset along the x dimension during the pick. We expect some additional grasp pose noise due to position error during grasp and object shift during grasp. This led to a randomized in-hand offset while running the trained insertion policy. Our results show that GoFlow can indeed more robustly generalize to real-world robot settings under pose uncertainty.

	FullDR	NoDR	DORAEMON	LSDR	ADR	GoFlow
Success Rate	6/10	3/10	5/10	5/10	5/10	**9/10**

Table 2: Real-world experimental results with the statistically significant results bolded.

A.6 Statistical Tests

We performed a statistical analysis of the simulated results reported in Figure 3 and the real-world experiments in Table 2. For the simulated results, we recorded the final domain coverages across all seeds and performed pairwise t-tests between each method and the top-performing method. The final performance mean and standard deviation are reported in Table 1. Any methods that were not significantly different from the top performing method ($p < 0.05$) are bolded. This same method was used to test significance of the real-world results.

A.7 Derivation of the Reward-Weighted Flow Gradient

In one component of our optimization, we aim to maximize the expected reward:

$$\mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \right],$$

where $p_\phi(\xi)$ is parameterized by ϕ, which are the parameters of the normalizing flow sampling distribution. The expectation can be written as:

$$\mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \right] = \int p_\phi(\xi) J_\xi(\pi) \, d\xi.$$

Taking the gradient with respect to ϕ, we use the log-derivative trick:

$$\nabla_\phi p_\phi(\xi) = p_\phi(\xi) \nabla_\phi \log p_\phi(\xi).$$

Substituting this into the integral:

$$\nabla_\phi \mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \right] = \int p_\phi(\xi) \nabla_\phi \log p_\phi(\xi) J_\xi(\pi) \, d\xi.$$

Rewriting as an expectation:

$$\nabla_\phi \mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \right] = \mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \nabla_\phi \log p_\phi(\xi) \right].$$

In practice, this expectation is approximated using samples $\{\xi_i\}_{i=1}^B$ drawn from $p_\phi(\xi)$:

$$\nabla_\phi \mathbb{E}_{\xi \sim p_\phi(\xi)} \left[J_\xi(\pi) \right] \approx \frac{1}{B} \sum_{i=1}^{B} J_{\xi_i}(\pi) \nabla_\phi \log p_\phi(\xi_i).$$

This expression is used to update the parameters of our normalizing flow as shown in Line 9 of Algorithm 1.