RoCUS: Robot Controller Understanding via Sampling

Anonymous Author(s)
Affiliation
Address
email

Abstract: As robots are deployed in complex situations, engineers and end users must develop a holistic understanding of their behaviors, capabilities, and limitations. Some behaviors are directly optimized by an objective, but typically many are emergent. The behaviors which are most often directly optimized relate to task completion: success rate, completion time, or total energy consumption. Other behaviors like collision avoidance, trajectory smoothness, and motion legibility are equally important for safe and trustworthy deployment, but are less often explicitly optimized. Designing an objective which optimizes every aspect of robot behavior is hard, so we instead propose a method for systematic analysis of exhibited behaviors. We present RoCUS: a Bayesian sampling-based method to find situations that lead to robot trajectories which exhibit specified behaviors—for example, highly jerky motions. For a 2D navigation problem and a 7 degree-of-freedom (DoF) arm reaching problem, we use RoCUS to analyze and compare three controllers: a deep imitation learning (IL) or reinforcement learning (RL) model, a rapidly exploring random tree (RRT) planner, and a dynamical system (DS) formulation. We show how RoCUS can uncover insights to further our understanding about these controllers beyond task-completion aspects and lead to better controller design. Code and video are submitted as supplementary files.

Keywords: Debugging and Evaluation, Algorithmic Transparency

1 Introduction

In 2018, after a confluence of failures, an autonomous vehicle (AV) struck and killed a pedestrian for the first time. In the run-up to this fateful event, the responsible company had reportedly been trying to improve the AV "ride experience" by emphasizing non-critical behaviors—such as the smoothness of the ride [1]. This event reflects the long-standing challenge in robotics of designing an appropriate objective which considers both safety-critical and non-critical behaviors. When crafting an objective, it is virtually impossible to proactively account for all potential controller behaviors, and some priorities may even be in conflict with one another [2]. In practice, any given robot behaviors may be specified, unspecified, or even misspecified [3], so extensive testing and evaluation is a critical component of designing and assessing robot controllers—especially learned controllers.

A common form of testing focuses on finding extreme and edge cases of controller failure. For example, a tester might use this procedure to find that the AV swerves very badly when encountering a farm animal while traveling at 60mph. Finding such extreme and edge cases is well-studied within both traditional software testing paradigms [4] and more recent adversarial perturbation testing methods [5]. However, we argue that an equally, if not more, important form should focus on *representative* scenarios, which considers the likelihood of encountering these scenarios. For example, if this AV is going to be deployed exclusively in New York City, the above example is largely unhelpful: cars rarely travel at 60mph in the city, and are very unlikely to encounter farm animals. Instead, the tester may prefer to know that the car swerves—though not as substantively—at lower speeds when a pedestrian steps toward it. Finding representative scenarios is often overlooked, but is especially useful for robotics. This is the focus of this paper.

Explicit mathematical analysis of robot controllers is implausible given the high dimensionality of the configuration space and the potential black-box representation of a learned controller. With access to an environment simulator, though, a straightforward testing approach is to roll out the robotic controller on various environments (e.g. road conditions under different weather and con-

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

gestion, with or without farm animals or pedestrians, etc.), and analyze those rollouts that exhibit a specified behavior—like excessive swerving. However, with too few environments, we risk missing the condition(s) that triggers the target behavior most saliently. With too many environments, all the most salient rollouts would be close to the global maximum at the expense of diversity and coverage. For example, if a farm animal causes the most swerving, followed by a pedestrian and a dangling tree branch, using too few environments may only find the pedestrian and the tree branch while using too many would result in an exclusive focus on the farm animal. Neither case helps the human develop a correct mental model of the AV's behavior.

To address this, we introduce Robot Controller Understanding via Sampling (ROCUS), a method to enable systematic behavior inspection. Using a posterior sampling procedure, ROCUS finds scenarios that are both inherently likely and which elicit specified behaviors. Analyzing these scenarios and the resulting trajectories can help engineers, testers, and end users inform their understanding of robot behaviors, and iterate on algorithm development if undesirable behaviors are revealed.

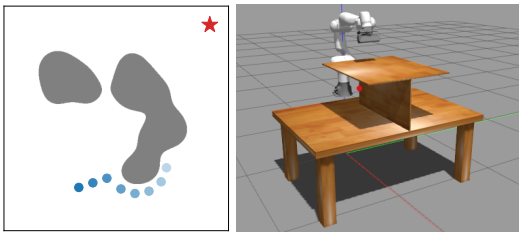

Figure 1: We analyze two problems using ROCUS. Left: 2D navigation tast, where the robot needs to reach the red star target while avoiding irregularly-shaped gray obstacles, e.g., by following the blue path. Right: 7DoF arm reaching task, where the robot needs to reach a given target (small red sphere on the left) which may be on either side under the T-shaped divider.

We use ROCUS to analyze three controllers on two common robotics tasks (Fig. 1). For a 2D navigation problem, we consider imitation learning (IL) [6], dynamical system (DS) [7], and rapidly-exploring random tree (RRT) [8]. For a 7DoF arm reaching problem, we consider reinforcement learning (RL) [9], as well as the same DS and RRT controllers. For each problem and controller, we specify several behaviors and visualize representative tasks and trajectories that elicit those behaviors. Through this analysis, we uncover insights that would be hard to derive analytically—insights which complement our mathematical understanding of the controllers. As such, ROCUS is a step towards the broader goal of building more accurate human mental models and enabling holistic evaluation of robot behaviors. Moreover, we include a case study on how to improve a controller based on new insights from ROCUS.

2 Related Work

Our work lies at the intersection of efforts to understand complex model behaviors and to benchmark robot performance. Methods to understand, interpret, and explain model behaviors are now commonplace in the machine learning community. [10] introduced Model Cards, a model analysis mechanism which breaks down model performance for data subsets. In natural language processing, [11] introduced a checklist for holistic evaluation of model capabilities and test case generation. [12] introduced BAYES-TREX, a Bayesian inference framework for sampling specified classifier behaviors. In robotics, [13] introduced a verification framework for assessing machine behavior by sampling parameter spaces to find temporal logic-satisfying behaviors. Other efforts aim to summarize robot policies, trading off factors like brevity, diversity, and completeness [14, 15]. All of these works have a shared underlying approach: treating the black box as immutable and performing downstream analyses of machine behavior [16]. ROCUS shares this approach, and, similar to [12], searches for instantiations which exhibit target behaviors to inform accurate human mental models.

While the need for benchmarking robot performance is often expressed [17, 18, 19], these efforts usually operate on distributions of trajectories or randomly selected trajectories, and the accompanying metrics are typically task-completion based without consideration of implicit performance factors. [20] put forth a recommendation of using *success weighted by path length* for navigation tasks—a task-completion metric. [21] and [22] introduced suites of metrics for comparing motion planning approaches, and [23] introduced a set of task and motion planning scenarios and metrics. Again, all of these proposed metrics are based solely on task completion. [24] introduced a set of performance measures for benchmarking reaching tasks; these benchmarks are either task-completion based or require a costly human motion ground truth. Our contribution is distinct in two ways. First, we propose to sample specific trajectories which communicate controller behaviors instead of reporting metrics averaged over distributions of trajectories. Second, we introduce metrics which draw on these prior works while also including essential alternative and typically emergent quality factors, like motion jerkiness and legibility [25].

3 RoCUS

RoCUS helps users understand robotic controllers via representative scenarios that exhibit various specified behaviors. A straightforward alternative to RoCUS is to simply run the controller on N different scenarios and pick the top-k with respect to the target behavior. While a bigger N leads to more salient behaviors in the top-k samples, these examples become more concentrated around the global maximum and less diverse, making this approach especially myopic and susceptible to multiple "local maxima" scenarios. Furthermore, it is not easy to find the optimal N to trade off between diversity and quality of the top-k samples.

RoCUS solves this by directly incorporating the distribution of scenarios—formally, *tasks*—into a Bayesian inference framework as shown in Fig. 2. A robotic problem is represented by a distribution $\pi(t)$ of individual tasks t. For example, a navigation problem may have $\pi(t)$ representing the distribution over target locations and obstacle configurations. Given a specific task t, the controller of interest induces a distribution $p(\tau|t)$ of possible trajectories τ. If both the controller and the transition dynamics are deterministic, $p(\tau|t)$ reduces to a δ-function at the induced trajectory τ. Stochasticity in either the controller (e.g., RRT) or the dynamics (e.g., uncertain outcome from an action) can result in τ being random. Finally, a behavior function $b(\tau, t)$ computes the behavior value of the trajectory—for example, the motion jerkiness. Some behaviors only depend on the trajectory and not the task, but we use $b(\tau, t)$ for consistency. Sec. 4 presents a list of behaviors.

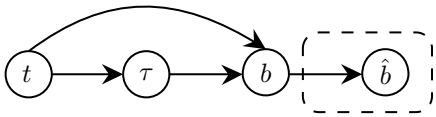

Figure 2: The graphical model for the inference problem of finding tasks t and trajectories τ which exhibit specific behaviors b. The dashed box indicates the relaxed formulation (Eq. 2).

The discussion on behavior in Sec. 1 is informal and implicitly combines two related but different concepts. The first concept is the behavior function $b(\tau, t)$ discussed above. The second is the specified target: for the swerving example, we are particularly interested in maximal behavior values. Thus, the target value can be thought of as $+\infty$. This inference problem uses the *maximal* mode of RoCUS. In other cases, we are also interested in tasks and trajectories that exhibit a specific finite behavior value. For example, we want to find road conditions that lead to a daily commute time of an hour, where the behavior is the travel time. This inference problem uses the *matching* mode.

Since matching mode is conceptually simpler, we present it first, followed by maximal mode. The sampling procedure is the same for both modes and presented last (Alg. 1).

3.1 Matching Mode

The exact objective is to find tasks and trajectories that exhibit user-specified behaviors b^*:

$$t, \tau \sim p(t, \tau | b = b^*) \propto p(b = b^* | t, \tau)\pi(\tau|t)\pi(t). \tag{1}$$

In most cases this posterior does not admit direct sampling, and an envelope distribution is not available for rejection sampling. Markov-Chain Monte-Carlo (MCMC) sampling does not work either: since the posterior is only non-zero on a very small or even measure-zero set, a Metropolis-Hastings (MH) sampler [26] can be stuck in the zero-density region.

Similar to [12], we relax the formulation as shown in Fig. 2:

$$\widehat{b}|b \sim \mathrm{No}(b, \sigma^2) \qquad t, \tau \sim p(t, \tau | \widehat{b} = b^*) \propto p(\widehat{b} = b^* | t, \tau) p(\tau|t)\pi(t). \tag{2}$$

This relaxed posterior is non-zero everywhere $\pi(t)$ is non-zero and provides useful guidance to an MH sampler. While σ is a hyper-parameter in [12], we instead assign σ such that

$$\int_{b^*-\sqrt{3}\sigma}^{b^*+\sqrt{3}\sigma} p(b)\,\mathrm{d}b = \alpha, \;\; \text{with} \;\; p(b) = \int_t \int_\tau p(\tau|t)\pi(t)\mathbb{1}_{b(\tau,t)=b}\,\mathrm{d}\tau\,\mathrm{d}t \tag{3}$$

being the marginal distribution of $b(\tau, t)$. Since we do not have a closed-form solution to $p(b)$, we can approximate it with its empirical count distribution, $\widehat{p}(b) = \sum_{i=1}^{N_B} \mathbb{1}_{b=b_i}/N_B$ for $\{b_1, ..., b_{N_B}\}$ drawn from the marginal distribution.

This formulation has two desirable properties. First, it is scale-invariant with respect to $b(\tau, t)$. For example, the posterior distribution is the same regardless of whether the trajectory length is measured in meters or centimeters. Second, the hyper-parameter $\alpha \in [0, 1]$ is intuitively interpreted as the approximate "volume" of posterior samples t, τ where $\widehat{b} = b^*$ under the marginal $p(t, \tau) = p(\tau|t)\pi(t)$ relieving the users of choosing σ manually. Details are derived in App. A.

3.2 Maximal Mode

In this mode, RoCUS finds trajectories that lead to maximal behavior values: $b^* \to \pm\infty$. It can also be used for finding minimal behavior values by negating the behavior. The posterior formulation is:

$$b_0 = \frac{b - \mathbb{E}[b]}{\sqrt{\mathbb{V}[b]}}, \quad \beta = \frac{1}{1 + e^{-b_0}}, \quad \widehat{\beta} \sim \mathrm{No}\left(\beta, \sigma^2\right), \quad t, \tau \sim p(t, \tau | \widehat{\beta} = 1), \tag{4}$$

where $\mathbb{E}[b]$ and $\mathbb{V}[b]$ are the mean and variance of the marginal $p(b)$. σ is chosen such that

$$\int_{1-\sqrt{3}\sigma}^{1} p(\beta)\, \mathrm{d}\beta = \alpha, \tag{5}$$

where $p(\beta)$ is the marginal distribution similar to Eq. 3. If $p(b)$ is normal, $p(\beta)$ is logit-normal. This formulation is again scale-invariant and has the same "volume" interpretation for α (App. A).

3.3 Posterior Sampling

To sample from the posterior, we first need to consider whether the controller and the dynamics are deterministic or stochastic.

Deterministic Controller and Dynamics: When both the controller and the dynamics are deterministic, so is $\tau|t$, denoted as $\tau(t)$. Eq. 2 reduces to

$$t \sim p(t|\widehat{b} = b^*) \propto p(\widehat{b} = b^*|t, \tau(t))\pi(t). \tag{6}$$

The reduction for Eq. 4 is similar.

Using MH for sampling is now viable, as presented in Alg. 1. First, σ is computed from α (Line 2). Then we start with an initial task t (Line 3). For each of the N iterations, we propose a new task t_{new} according to a transition kernel and compute the forward and reverse transition probabilities $p_{\text{for}}, p_{\text{rev}}$ (Line 6). We evaluate the posteriors under t and t_{new} (Line 6 and 7) and calculate the acceptance ratio using the MH detailed balance principle (Line 8). Finally, we accept or reject accordingly (Line 9 – 11). Note that if the proposal is rejected, the current t is left unchanged *and appended to the samples*. In the end, we can discard the first N_B samples as burn-in, and/or thin the samples by only keeping every N_T samples to reduce auto-correlation.

Algorithm 1: MH Sampling Procedure

Input: "Posterior volume" α, numner of samples N, optional burn-in N_B and thinning period N_T.

1 samples $\leftarrow$ [ ];
2 Get σ from α by Eq. 3 (matching) or 5 (maximal);
3 Randomly initialize t;
4 **for** $i = 1, ..., N$ **do**
5 $\quad t_{\text{new}}, p_{\text{for}}, p_{\text{rev}} = \text{propose}(t)$
6 $\quad$ Get p from t by Eq. 2 (match) or Eq. 4 (max)
7 $\quad$ Get p_{new} from t_{new} by Eq. 2 or Eq. 4;
8 $\quad a \leftarrow (p_{\text{new}} \cdot p_{\text{rev}})/(p \cdot p_{\text{for}})$;
9 $\quad$ Sample $u \sim \text{Unif}[0, 1]$;
10 $\quad$ **if** $u < a$ **then**
11 $\quad\quad t \leftarrow t_{\text{new}}$;
12 $\quad$ Append t to samples;
13 Optionally, discard the first N_B burn-in samples and thin the samples by only keeping every N_T samples;
14 **return** samples

Stochastic Controller: When the controller and $p(\tau|t)$ are stochastic, we need to sample in the combined (t, τ)-space. Typically the controller can be implemented by sampling a random variable u, and then producing the action based on the realization of u. For instance, for a Normal stochastic policy represented as action $a \sim \mathrm{No}(\mu(s), \sigma(s)^2)$ at state s, we can sample $u \sim \mathrm{No}(0, 1)$ and calculate $a = \mu(s) + u \cdot \sigma(s)$. Fig. 3 shows the graphical model diagram that explicitly accounts for the controller randomness with random variable u.

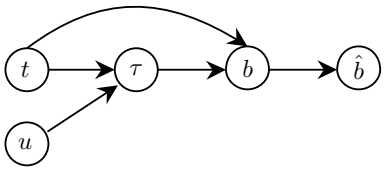

Figure 3: The same graphical model as in Fig. 2, but with the addition of stochasticity u in the controller such that $\tau|t, u$ is now deterministic.

Using this transformation, Eq. 2 can be developed as

$$p(t, \tau|\widehat{b} = b^*) \propto p(\widehat{b} = b^*|t, \tau(e, u))p(u|t)\pi(t), \tag{7}$$

where we overload $\tau(t, u)$ to refer to the *deterministic* trajectory given the task t and controller randomness u. It is crucial that we can evaluate $p(u|t)$ for any u. Typically, we have independence $u \perp t$, so $p(u|t) = p(u)$. Concretely, building upon Alg. 1, u_{new} is proposed alongside with t_{new} (Line 5), the detailed balancing principle (Line 8) is multiplied by $p_{u,\text{rev}}/p_{u,\text{for}}$, and $t_{\text{new}}, u_{\text{new}}$ are accepted or rejected together (Line 10 – 12).

Stochastic Dynamics: Using the same logic, the above framework can also accommodate stochasticity in transition dynamics (e.g. object position uncertainty after it is pushed), *as long as such stochasticity can be captured in a random variable v and $p(v|t)$ can be evaluated.* This is typically possible in simulation, and the modification to Alg. 1 is similar to the previous case. In the real world, we can instead treat a sampled trajectory as the deterministic one, restart multiple times to estimate $\mathbb{E}_\tau[b(\tau, t)]$, and use likelihood-free MCMC methods [27]. We leave these investigations to future work, and use deterministic dynamics in our experiments.

RoCUS uses Bayesian sampling concepts of prior, likelihood, and posterior quite liberally. Specifically, the task distribution is defined as the prior, and thus the notion of a task being likely in the deployment context refers to high probability under the prior. Likelihood refers to the behavior saliency: how much the exhibited behavior matches the target value or has large magnitude. The act of posterior sampling then finds tasks that strike a balance between these two objectives.

The choice of explicitly modeling the task distribution is intentional, as it is not unlikely that the experienced deployment environment will be different than the development environment. Such a domain mismatch may cause catastrophic failures, especially for learned controllers whose extrapolation behaviors are typically undefined. With a suitable task distribution, RoCUS allows more failures to surface during this testing procedure.

4 Behavior Taxonomy

Robot behaviors broadly belong to one of two classes: intentional and emergent. Behaviors that are specified as objectives for a controller are *intentional*. For example, in a reaching problem a controller likely optimizes to move the end-effector to the target. This optimization occurs by setting the target as an attractor in a DS, setting the objective configuration so the target is reached in RRT, or rewarding proximity in RL. Final distance between the end-effector and the target is an intentional behavior for all three controllers. In comparison, *emergent* behaviors are not explicitly specified in the objective. For this same reaching problem, an RL policy with reward based solely on distance may exhibit smooth trajectories for some target locations and jerky, abrupt trajectories for others. Such unspecified behaviors may emerge due to robot kinematic structure, training stochasticity, or model inductive bias.

While many behaviors are specific to robots or tasks, some are widely applicable to different robots (e.g., arm, humanoid, or soft robots). Roboticists applying these metrics must choose an appropriate subset and specific instantiations for their robot morphology and task. Given a trajectory τ, most behavior metrics $b(\tau, t)$ can be expressed as a line integral of a scalar field $V(\mathbf{x})$ along τ or its length-normalized integral,

$$b = \int_\tau V(\mathbf{x})\,\mathrm{d}s \quad \text{or} \quad b = \frac{1}{||\tau||}\int_\tau V(\mathbf{x})\,\mathrm{d}s, \tag{8}$$

where $\mathrm{d}s$ is the infinitesimal segment on τ at $\mathbf{x}$ and $||\tau||$ is the trajectory length. $\mathbf{x}$ and τ can be in either joint space or task space. We numerically compute it via Riemann integration.

We consider five classes of behaviors; users can add alternative behaviors for use with RoCUS.

Length: the distance traveled during a trajectory. We set $V(\mathbf{x}) = 1$ in the unnormalized integral.

Time Derivatives: the first, second, and third time derivatives of the trajectory correspond to the velocity, acceleration, and jerk. We have $V = ||\dot{\mathbf{x}}||, ||\ddot{\mathbf{x}}||$ or $||\dddot{\mathbf{x}}||$. It is typical to study the mean velocity (i.e. the normalized integral), but both total and mean acceleration/jerk are also of interest.

Straight-Line Deviation: the average deviation of the trajectory from a straight line connection, which may or may not be collision-free, in the task or joint-space. We have $V(\mathbf{x}) = ||\mathbf{x} - \mathrm{proj}_{\mathbf{x}^* - \mathbf{x}_0}\mathbf{x}||$, where $\mathbf{x}_0$ and $\mathbf{x}^*$ are the start and target configurations, and $\mathrm{proj}_{\cdot}(\cdot)$ returns the projection of $\mathbf{x}$ onto this straight line. We use the normalized integral.

Obstacle Clearance: the average distance to the closest obstacle, from the end effector or the whole body. We have $V(\mathbf{x}) = \min_{\mathbf{x}_o \in \mathcal{O}} ||\mathbf{x} - \mathbf{x}_o||$, for obstacle space $\mathcal{O}$. We use the normalized integral.

Motion Legibility: how well the goal can be predicted over the course of the exhibited trajectory. A generic choice for is $V(\mathbf{x}) = p(g|\mathbf{x})$, or the conditional probability of the goal g given at the current robot state $\mathbf{x}$, although there may be better application-specific definitions. We then use the normalized integral to measure the average legibility.

5 RoCUS Use Case Demos

In this section, we concretely demonstrate how RoCUS can be used to find "hidden" properties of various controllers for two common tasks, navigation and reaching. Furthermore, during the analyses we uncovered a suboptimal controller design due to bad hyper-parameter choices, and findings by RoCUS point out a remedy option, which results in a much better controller.

5.1 Controller Algorithms

In total, we consider four classes of robot controllers. The **imitation learning** (IL) controller uses expert demonstrations to learn a neural network policy which maps observations to deterministic actions. The **reinforcement learning** (RL) controller implements proximal policy gradient (PPO) [28]. While a mean and a variance is used to parameterize a PPO policy, at execution time, the policy deterministically outputs the mean action. The **dynamical system** (DS) controller modulates the linear controller $\mathbf{u}(\mathbf{x}) = \mathbf{x}^* - \mathbf{x}$, for the task-space target $\mathbf{x}^*$, using the modulation matrix M proposed by [7]: $\mathbf{u}_M(\mathbf{x}) = M \cdot \mathbf{u}(\mathbf{x})$. M is calculated from a list of monotonically increasing Γ-functions, one per obstacle. We give a self-contained review of this approach in App. B.

The **rapidly-exploring random tree** (RRT) controller finds a configuration-space trajectory via RRT and then controls the robot through descretized segments. There are many RRT variants with subtle differences. For clarity, Algorithm 2 presents the version that we use.

Algorithm 2: RRT Algorithm

Input: Start configuration s_0, target configuration s^*.

$\mathcal{T} \leftarrow$ tree(root = s_0);
success $\leftarrow$ attempt-grow($\mathcal{T}$, from = s_0, to = s^*);
while *not* success **do**
  $s \leftarrow$ sample-configuration( );
  $s_n \leftarrow$ nearest-neighbor($\mathcal{T}$, s);
  success $\leftarrow$ attempt-grow($\mathcal{T}$, from = s_n, to = s);
  if success **then**
    success $\leftarrow$ attempt-grow($\mathcal{T}$, from = s, to = s^*);
return path($\mathcal{T}$, from = s_0, to = s^*)

While RRT is stochastic (unlike DS, IL and RL), the entire randomness is captured by the sequence of C-space samples used to grow the tree, including failed ones. We call this a *growth* $g = [s_1, s_2, s_3, ...]$. The probabilistic completeness property of RRT generally assures that the algorithm will terminate in finite time with probability 1 if a path to the target exists [8]. Thus, hypothetically, given an infinitely long tape containing every entry of g, we can compute a deterministic trajectory $\tau = \mathrm{RRT}(s_0, s^*, g)$ with a finite number of nodes with probability 1.

To enable MH inference, we take inspiration from Bayesian nonparametrics: we instantiate g on an *as-needed* basis. We start with an empty vector of $g = []$. When calculating $\mathrm{RRT}(s_0, s^*, g)$, if a new point beyond existing entries of g needs to be sampled, we append it to g. During MH inference, we use a transition kernel that operates element-wise on instantiated entries of g (i.e. independently perturbing each entry of g). If the transition kernel does not depend on the current g (e.g. drawing uniformly from the C-space), then past instantiated entries do not even need to be kept.

Note that RRT trajectories are often smoothed *post hoc*. Since our main focus is to evaluate and identify problems for an existing one, we use the original formulation. Moreover, it is easy to use RoCUS to evaluate model updates (e.g. original vs smoothed RRT) as discussed in Sec. 6.

The MCMC sampling uses a Gaussian drift kernel, and additional details are in App. C.

5.2 2D Navigation Task Experiments

Setup In a rectangular arena with irregularly shaped obstacles, a point mass robot needs to move from the lower left to the upper right corner (Fig. 1 left). App. D details the obstacle generation and robot simulation procedures and contains more environment visualizations.

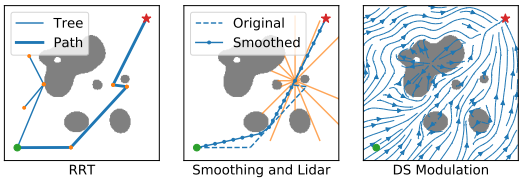

Figure 4: RRT, IL and DS controllers on 2D navigation domain. Left: the RRT controller tree. Middle: smoothed RRT trajectory and lidar sensor (orange lines) for IL controller training. Right: the modulation by the DS controller.

We consider three controllers for this environment: an RRT planner, a deep learning IL policy, and a DS (Fig. 4). The RRT planner implements Alg. 2 and discretizes the path to small segments as control signals at each time step. The IL controller uses smoothed RRT trajectories as expert demonstrations, and learns to predict heading angle from its current position and lidar readings. The DS controller finds an interior reference point for each obstacle, and converts each

obstacle in the environment to be star-shaped. Γ-functions are then defined for these obstacles and used to compute the modulation matrix M. App. E contains additional implementation details.

Straight-Line Deviation In most cases, the robot cannot navigate directly to the target in a straight line. Thus, the collision-avoidance behavior is a crucial aspect for for navigation robots. To understand it, we sample obstacles that lead to trajectories minimally deviating from the straight line path. Since the deviation is always non-negative, we use the match-type posterior in Eq. 2 with target $b^* = 0$. In Fig. 5, the top row plots posterior trajectories in orange, with prior trajectories in blue, for three controllers. The bottom row plots the obstacle distributions compared to the prior, with red regions being more likely to be occupied by obstacles and blue ones less likely to be obstructed.

The visualization shows that for DS and RRT, the posterior trajectories and obstacle configurations are mostly symmetric with respect to the straight-line connection, as expected since both methods are formulated symmetrically with respect to the x- and y-coordinates.

For the obstacle distribution, the RRT result is expected. Since RRT seeks straight-line connections whenever possible, the posterior largely favor a "diagonal corridor" with obstacles on either side. For the DS, obstacles are slightly *more* likely to exist at the two ends of the above-mentioned corridor. This behavior is an artifact of the DS *tail effect*, which drags the robot around the obstacle (details in App. B). By taking advantage of anchor-like obstacles at the ends of the corridor, the modulation can reliably minimize the straight-line deviation.

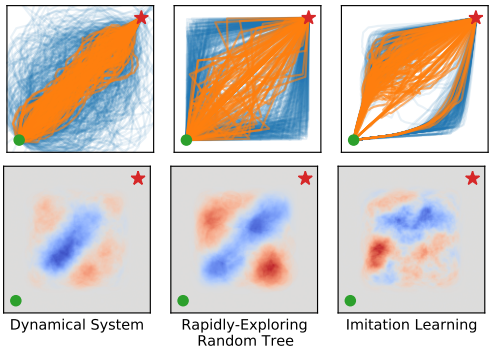

Figure 5: Top: Posterior trajectories in orange vs. prior in blue for minimal straight-line deviation behavior. Bottom: Posterior obstacle distribution, with red for higher obstacle density and blue for lower. IL is notably asymmetric, and DS notably has obstacles obscuring the direct path.

The IL controller saliently exhibits trajectory asymmetry: it mostly takes paths on the left of the robot. It is possible that the asymmetry is due to "unlucky" samples by the MH sampler, but many independent restarts all confirm the asymmetry, indicating that the asymmetry is inherent in the learned model. Since the neural network architecture is symmetric, we conclude that the stochasticity in the dataset generation and training procedure (e.g. initialization) leads to such imbalanced behaviors. Furthermore, the obstacle map suggests that obstacles are distributed very close to the robot path. Why does the robot seem to drive into obstacles? The answer lies in dataset generation process: the smoothing procedure (Fig. 4 middle) results in most demonstrated paths navigating tightly around obstacles, and it is therefore natural to expect the learned IL controller to display the same behavior.

Takeaways ROCUS reveals two unexpected phenomena. First, IL trajectories are highly asymmetric toward the left of the obstacle due to dataset and/or training imbalance. Second, both DS and IL models exhibit certain "obstacle-seeking" behaviors, the former due to the "tail-effect" and the latter due the dataset generation process. In both cases, such behavior may be undesirable in deployment due to possibly imprecise actuation, and the controller design may need to be modified. We also studied the legibility and obstacle clearance of these controllers, with results in App. F.

5.3 7DoF Arm Reaching Task Experiments

Setup A 7DoF Franka Panda arm is mounted on the side of a table with a T-shaped divider (Fig. 1 right). Starting from the same initial configuration on top of the table, it needs to reach a random location on either side under the divider. We simulate this task in PyBullet [29]. We again consider three controllers: an RRT planner, a deep RL PPO agent, and a DS formulation.

RRT again implements Algorithm 2, but uses inverse kinematics (IK) to first find the joint configuration corresponding to the target location. The RL controller is a multi-layer perceptron (MLP) network trained using the PPO algorithm. The DS model outputs the end-effector trajectory in the task space, which is converted to joint space via IK, with SVM-learned obstacle definitions. Overall, RRT and RL are quite successful in reaching the target while the DS is not due to the bulky robot structure, close proximity to the divider, and the task-space only modulation. App. G contains additional implementation details for each method.

End-Effector Movement We find configurations that minimize the total travel distance of the end-effector for RRT and RL (DS omitted due to high failure rate). Fig. 6 (left two) shows the posterior

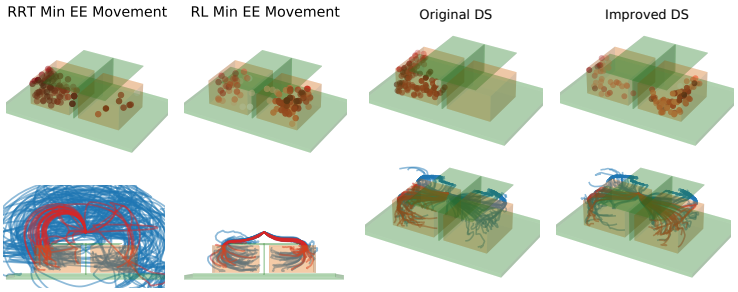

Figure 6: Left: Minimal end-effector movement samples for RRT and RL. Right: Posterior samples for minimal distance from end-effector to target for the original and improved DS controllers. Top: posterior targets locations, with tabletop + divider in green and target region in orange. Bottom: posterior trajectories in red, prior trajectories in blue. Robot is mounted on the near long edge.

target locations and trajectories. Notably, unlike RL, RRT trajectories are highly asymmetric, since there are straight-line connections in the configuration space from the initial pose to some target regions on the left, while every right-side goal requires at least an intermediate node.

DS Improvement with RoCUS Our initial DS implementation frequently failed to reach the target. This is understandable, as the DS convergence guarantee [7] is only valid in task space, in which we defined the modulation. When the full-arm motion is solved via IK, it is possible that some body parts may collide and get stuck because of the table divider. To understand the DS behaviors, we used RoCUS to sample target locations that result in minimal final distance from the end-effector to the target (i.e., most successful executions, Fig. 6 center-right). Similar to the RRT case, the samples show strong lateral asymmetry, with all posterior target locations on the left, due to the same cause of asymmetric kinematic structure. The result points to a clear path to improve the DS controller such that it can succeed with right-side targets: increase the collision clearance of the divider so that the end-effector navigates farther away from the divider, thus also bringing the whole arm to be farther away. As detailed in App. H.1, this modification greatly improves the controller performance as confirmed by the new symmetry in Fig. 6 (rightmost).

Takeaway The set of studies reveal an important implication of the robot's kinematic structure: the left side is much less "congested" with obstacles than the right side in the configuration space. While the RL method is able to learn efficient policies for both sides, the design of certain controllers may need to explicitly consider such factors. An additional study on legibility is in App. H.

6 Discussion and Future Work

RoCUS enables humans to build better mental models of robot controllers. It complements existing task-completion-centric evaluations on hand-designed scenarios by generating scenarios and trajectories for specified behaviors in a principled way. We demonstrated how it can uncover non-obvious insights in two domains, and showed how it can help with debugging and improving a controller.

While we only used RoCUS on individual controllers, future work can readily extend it to *compare two controllers* by defining behavior functions that take in the task and two trajectories, one from each controller, and compute differential statistics. For example, this could be used to find road conditions that lead to increased swerving behavior of a new AV controller, compared to the existing one. Such testing is important to gain a better understanding of *model updates* [30], and is particularly necessary for ensuring that these updates do not unintentionally introduce new problems.

In addition, sometimes it is important to understand particular trajectories sampled by RoCUS. For example, why does the controller take one action rather than another (e.g. swerving rather than braking)? Preliminary investigation into this explainable artificial intelligence (XAI) problem in the context of temporally extended decision making has been undertaken [31, 32], but [33] identifies various problems in current explanations and future research is very much needed to address them.

Finally, an important step before actual deployment is to design appropriate user interfaces to facilitate the two-way communication between RoCUS and end-users. In one direction, the user needs to specify the behavior of interest, and it would be desirable for it to involve as little programming as possible, especially for non-technical stakeholders. In the other direction, RoCUS needs to present the sample visualization, and potentially model explanations as described above, for users to inspect. Here, it is important for the information to be accurate but at the same time not overwhelming.

Overall, RoCUS is a framework for systematic discovery and inspection of representative robotic controller scenarios. In general, we hope the demonstrated usefulness of tools like RoCUS sparks further efforts towards developing support for more holistic understanding of robot controllers.

References

[1] J. Bort. Inside Uber before its self-driving car killed a pedestrian: Sources describe infighting, 'perverse' incentives, and questionable decisions, 2018. URL https://www.businessinsider.com/sources-describe-questionable-decisions-and-dysfunction-inside-ubers-self-driving-unit-before-one-of-its-cars-killed-a-pedestrian-2018-10.

[2] A. Rasouli and J. K. Tsotsos. Autonomous vehicles that interact with pedestrians: A survey of theory and practice. *IEEE transactions on intelligent transportation systems*, 21(3), 2019.

[3] A. Bobu, A. Bajcsy, J. F. Fisac, S. Deglurkar, and A. D. Dragan. Quantifying hypothesis space misspecification in learning from human–robot demonstrations and physical corrections. *IEEE Transactions on Robotics*, 36(3):835–854, 2020.

[4] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler. *The art of software testing*, volume 2. Wiley Online Library, 2004.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[6] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *RAS*, 57(5):469–483, 2009.

[7] L. Huber, A. Billard, and J.-J. Slotine. Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters*, 2019.

[8] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[9] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[10] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.

[11] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. *arXiv preprint arXiv:2005.04118*, 2020.

[12] S. Booth, Y. Zhou, A. Shah, and J. Shah. Bayes-TrEx: a bayesian sampling approach to model transparency by example. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[13] C. Fan, X. Qin, and J. Deshmukh. Parameter searching and partition with probabilistic coverage guarantees. *arXiv preprint arXiv:2004.00279*, 2020.

[14] B. Hayes and J. A. Shah. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pages 303–312. IEEE, 2017.

[15] I. Lage, D. Lifschitz, F. Doshi-Velez, and O. Amir. Exploring computational user models for agent policy summarization. *arXiv preprint arXiv:1905.13271*, 2019.

[16] I. Rahwan, M. Cebrian, N. Obradovich, J. Bongard, J.-F. Bonnefon, C. Breazeal, J. W. Crandall, N. A. Christakis, I. D. Couzin, M. O. Jackson, et al. Machine behaviour. *Nature*, 2019.

[17] J. Mahler, R. Platt, A. Rodriguez, M. Ciocarlie, A. Dollar, R. Detry, M. A. Roa, H. Yanco, A. Norton, J. Falco, et al. Guest editorial open discussion of robot grasping benchmarks, protocols, and metrics. *TASE*, 15(4):1440–1442, 2018.

[18] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. 2019.

[19] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[20] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. 2018.

[21] B. Cohen, I. A. Şucan, and S. Chitta. A generic infrastructure for benchmarking motion planners. In *IROS*, pages 589–595. IEEE, 2012.

[22] M. Moll, I. A. Sucan, and L. E. Kavraki. Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization. *IEEE RAM*, 22(3):96–102, 2015.

[23] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki. Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters*, 3(4):3765–3772, 2018.

[24] A. Lemme, Y. Meirovitch, M. Khansari-Zadeh, T. Flash, A. Billard, and J. J. Steil. Open-source benchmarking for learned reaching motion generation in robotics. *Paladyn, Journal of Behavioral Robotics*, 1(open-issue), 2015.

[25] A. D. Dragan, K. C. Lee, and S. S. Srinivasa. Legibility and predictability of robot motion. In *HRI*. IEEE, 2013.

[26] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[27] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[30] G. Bansal, B. Nushi, E. Kamar, D. S. Weld, W. S. Lasecki, and E. Horvitz. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *AAAI*, 2019.

[31] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding atari agents. In *ICML*, pages 1792–1801. PMLR, 2018.

[32] T. Zahavy, N. Ben-Zrihem, and S. Mannor. Graying the black box: Understanding dqns. In *ICML*, pages 1899–1908. PMLR, 2016.

[33] A. Atrey, K. Clary, and D. Jensen. Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning. *arXiv preprint arXiv:1912.05743*, 2019.

[34] S. M. Khansari-Zadeh and A. Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 32(4):433–454, 2012.

[35] Y. Zhou and K. Hauser. 6DOF grasp planning by optimizing a deep learning scoring function. In *RSS Workshop on Revisiting Contact – Turning a Problem into a Solution*, 2017.

[36] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. In *CVPR*, pages 2174–2182, 2017.

[37] K. Hauser. Robust contact generation for robot simulation with unstructured meshes. In *IJRR*. Springer, 2016.

[38] S. S. Mirrazavi Salehian, N. Figueroa, and A. Billard. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232, 2018.