RoCUS: Robot Controller Understanding via Sampling

Yilun Zhou[1], Serena Booth[1], Nadia Figueroa[1], and Julie Shah[1]

***Abstract*— As robots are deployed in complex situations, engineers and end users must develop a holistic understanding of their capabilities and behaviors. Existing research focuses mainly on factors related to task completion, such as success rate, completion time, or total energy consumption. Other factors like collision avoidance behavior, trajectory smoothness, and motion legibility are equally or *more* important for safe and trustworthy deployment. While methods exist to analyze these quality factors for individual trajectories or distributions of trajectories, these statistics may be insufficient to develop a mental model of the controller's behaviors, especially uncommon behaviors. We present RoCUS: a Bayesian sampling-based method to find situations that lead to trajectories which exhibit certain behaviors. By analyzing these situations and trajectories, we can gain important insights into the controller that are easily missed in standard task-completion evaluations. On a 2D navigation problem and a 7 degree-of-freedom (DoF) arm reaching problem, we analyze three controllers: a rapidly exploring random tree (RRT) planner, a dynamical system (DS) formulation, and a deep imitation learning (IL) or reinforcement learning (RL) model. We show how RoCUS can uncover insights to further our understanding about them beyond task-completion aspects. The code is available at `https://github.com/YilunZhou/RoCUS`.**

I. INTRODUCTION

A common goal in robotics is to remove safety restraints such as protective screens or fences between humans and robots [1]. This allows robots to more effectively participate in human environments. One challenge is that humans in close proximity to a robot must be able to anticipate the robot's behaviors, particularly any undesirable behaviors. Meanwhile, most robotic algorithms optimize a single objective. For example, RRT tries to reach a specific target configuration, and RL maximizes the cumulative discounted reward. With ingenuity, a cost or reward function can include multiple facets of target behaviors, like obstacle avoidance and task completion. Yet, anticipating the consequences of a multifaceted cost function is challenging, and learned agents are infamous for finding and exploiting loopholes, or "reward hacking" [2]. Thus, it is important to understand the robot's emergent behaviors—any of which may be specified, unspecified, or even misspecified [3] in the objective function.

One way to understand robot behaviors is through mathematical analyses, e.g., finding convergence bounds or proving behavior constraints. However, such analyses are rarely applied beyond simple cases due to the high dimensionality of the configuration space or the black-box representation of a learned controller. An alternative approach is to observe the robot in action. While this is straightforward, it can be time consuming and tedious, and can lead to misinformed human mental models when a behavior occurs infrequently. We propose a method for systematic behavior inspection called Robot Controller Understanding via Sampling (RoCUS). Using a posterior sampling procedure, our method finds scenarios that elicit trajectories with specified behaviors. This can be hard, so our method relaxes the specified behavior constraints to guide sampling. Analyzing these scenarios and the resulting trajectories lets engineers and users broaden their understanding of robot behaviors, and iterate algorithm development if undesirable behaviors are revealed.

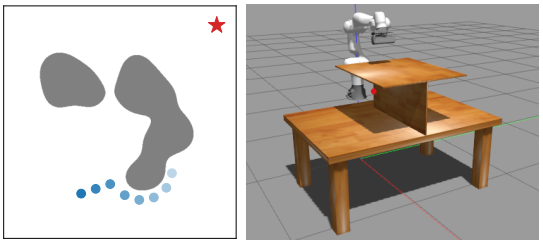

Fig. 1. We analyze two problems using RoCUS. Left: 2D navigation, where the robot needs to reach the red star target while avoiding irregularly-shaped gray obstacles. Right: 7DoF arm reaching, where the robot needs to reach a given target (red sphere) on either side under the T-shaped divider.

The idea of finding specific examples for model assessment has been proposed in natural language processing (NLP) [4], in a constrained procedural autonomous vehicle simulator [5], and for general machine learning (ML) classification [6]. In robotics, many metrics have been proposed to benchmark robot performance [7], [8], [9], [10], [11]. Currently, these proposals operate on randomly selected individual trajectories or distributions of trajectories, and the metrics focus primarily on task completion. RoCUS instead finds trajectories that satisfy specified behavior metric targets. Further, we propose a suite of metrics to evaluate emergent robot behaviors, including motion legibility [12] and obstacle avoidance that are often overlooked.

We use RoCUS to analyze three controllers on two problems (Fig. 1). For a 2D navigation problem, we consider imitation learning (IL) [13], dynamical system (DS) [14], and rapidly-exploring random tree (RRT) [15]. For a 7DoF robotic arm reaching problem, we consider reinforcement learning (RL) [16], as well as the same DS and RRT controllers but with inverse kinematics. For each problem and controller, we consider several behaviors and visualize representative tasks and trajectories that elicit those behaviors. Through visual analysis, we uncover insights that would be hard to derive analytically and so complement our mathematical understanding of the controllers. As such, RoCUS is a step toward achieving the broader goal of more accurate human mental models of robot behavior.

[1]All authors are affiliated with MIT Computer Science and Artificial Intelligence Laboratory. Correspondence: {yilun, serenabooth, nadiafig, julie_a_shah}@csail.mit.edu

II. RELATED WORK

Our work lies at the intersection of efforts to understand behaviors generated by complex models and to benchmark robot performance. Methods to understand, interpret, and explain model behaviors are now commonplace in the machine learning community. [17] introduced Model Cards, a model analysis mechanism which breaks down model performance for data subsets, enabling methodical assessment of model fairness and transparency. In NLP, [4] introduced a checklist for holistic evaluation of model capabilities and easy test case generation. Our prior work [6] introduces BAYES-TREX, a framework for sampling specified classifier behaviors using Bayesian inference. In robotics, [18] introduced a verification framework for assessing machine behavior by sampling parameter spaces to find temporal logic-satisfying behaviors. All of these frameworks have a shared underlying approach: analytical analysis of black box models can be intractable, so these methods treat the black box as immutable and perform downstream analyses of machine behavior [19].

While the need for benchmarking robot performance is often expressed [20], [21], [22], 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. [11] put forth a recommendation of using *success weighted by path length* as the principal evaluation metric for navigation tasks—a task-completion metric. [7] and [9] introduced suites of metrics for comparing motion planning approaches, and [10] introduced a set of task and motion planning scenarios and metrics. Again, all of these proposed metrics are based solely on task completion. [8] 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 sampling specific trajectories which communicate controller behaviors instead of reporting metrics averaged over distributions of trajectories. Second, we introduce a set of metrics which draw on these prior works while also including essential alternative quality factors, like motion jerkiness and legibility [12].

III. ROCUS

The goal of ROCUS is to help understand robotic controllers via representative scenarios that exhibit various behaviors. A straightforward way 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, they also become less diverse and more concentrated around the global maximum. This is especially troublesome when there are multiple "local maxima" scenarios and we want to understand all of them. Furthermore, it is not easy to find the optimal N to trade off between diversity and quality among 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 (without the dashed box). 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., an RRT planner) or the dynamics (e.g., uncertain effect of the same pushing force) can result in τ being random. Finally, a behavior function $b(\tau, t)$ computes the behavior value of the trajectory. Some behaviors only depend on the trajectory and not the task, but we use $b(\tau, t)$ for consistency. We present a list of behaviors in Section IV.

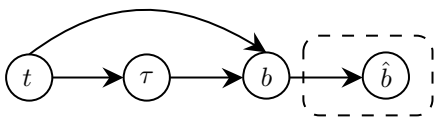

Fig. 2. The graphical model for the inference problem of finding tasks t and trajectories τ which exhibit specific behaviors b. The dashed box indicates the relaxed formulation (Eq. 3).

There are two types of inference, matching and maximal. In **matching** mode, ROCUS finds tasks and trajectories that exhibit user-specified behaviors b^*:

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

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 [23] can be stuck in the zero-density region.

Similar to [6], we relax the formulation as shown in Fig. 2:

$$\hat{b} | b \sim \mathrm{No}(b, \sigma^2), \quad (2)$$

$$t, \tau \sim p(t, \tau | \hat{b} = b^*) \propto p(\hat{b} = b^* | t, \tau) p(\tau | t) \pi(t). \quad (3)$$

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 [6], we instead assign σ such that

$$\int_{b^* - \sqrt{3}\sigma}^{b^* + \sqrt{3}\sigma} p(b) \, \mathrm{d}b = \alpha, \quad (4)$$

where

$$p(b) = \int_t \int_\tau p(\tau | t) \pi(t) \mathbb{1}_{b(\tau, t) = b} \, \mathrm{d}\tau \, \mathrm{d}t \quad (5)$$

is 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, $\hat{p}(b) = \sum_{i=1}^{N_B} \mathbb{1}_{b = b_i} / N_B$ 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 $b(\tau, t)$ (e.g. the trajectory length measured in meters vs. centimeters). In addition, the hyper-parameter $\alpha \in [0, 1]$ is intuitively interpreted as the approximate "volume" of posterior samples t, τ where $\hat{b} = b^*$ under the marginal $p(t, \tau) = p(\tau | t) \pi(t)$, relieving the users of choosing σ manually. Details are derived in Supp. A.

In **maximal** mode, ROCUS finds trajectories that lead to maximal target behaviors: $b^* \to \pm\infty$. Use cases include, for example, finding trajectories that maximize collision risk. This mode can also be used for finding minimized target behaviors. ROCUS uses the following posterior formulation:

$$b_0 = \frac{b - \mathbb{E}[b]}{\sqrt{\mathbb{V}[b]}}, \qquad \beta = \frac{1}{1 + e^{-b_0}}, \tag{6}$$

$$\hat{\beta} \sim \mathrm{No}\left(\beta, \sigma^2\right), \quad t, \tau \sim p(t, \tau | \hat{\beta} = 1), \tag{7}$$

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

$$\int_{1-\sqrt{3}\sigma}^{1} p(\beta)\, \mathrm{d}\beta = \alpha, \tag{8}$$

where $p(\beta)$ is the marginal distribution similar to Eq. 5. If $p(b)$ is normal, then $p(\beta)$ is logit-normal. This formulation is again scale-invariant and has the same "volume" interpretation for α (Supp. A).

To sample from the posterior, we need to consider whether the controller and/or the dynamics are deterministic or stochastic, and discuss the following three cases.

Deterministic Controller and Dynamics: When both the controller and the dynamics are deterministic, so is $\tau|t$, denoted as $\tau(t)$. Eq. 3 reduces to

$$t \sim p(t|\hat{b} = b^*) \propto p(\hat{b} = b^*|t, \tau(t))\pi(t). \tag{9}$$

The reduction for Eq. 7 is similar.

Using MH for sampling is now viable. We start with an initial task t. For each iteration, we propose a new task according to a transition kernel. We evaluate the proposed posterior of the new task with t and $\tau(t)$, and calculate the acceptance ratio using the MH detailed balance principle. Finally, we accept or reject accordingly. We repeat this procedure for N iterations, where N is a hyper-parameter.

Stochastic Controller: When the controller and $p(\tau|t)$ are stochastic, we need to explicitly 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 $a = \mu(s) + u \cdot \sigma(s)$.

Using this transformation, Eq. 3 can be developed as

$$p(t, \tau|\hat{b} = b^*) \propto p(\hat{b} = b^*|t, \tau(e, u))p(u|t)\pi(t), \tag{10}$$

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 $p(u|t) = p(u)$.

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. In the real world, we can instead:

1) treat a sampled trajectory as the deterministic one; or
2) restart multiple times to estimate $\mathbb{E}_\tau[b(\tau, t)]$; or
3) use likelihood-free MCMC methods [24].

We leave the study of these adaptations to future work, and use deterministic dynamics in our experiments.

IV. BEHAVIOR TAXONOMY

Robot behaviors generally 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. Thus, the 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 function. 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 other targets. 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 can be expressed as a line integral of a scalar field $V(\mathbf{x})$ along τ or its length-normalized integral.

$$\int_\tau V(\mathbf{x})\, \mathrm{d}s \quad \text{or} \quad \frac{1}{||\tau||} \int_\tau V(\mathbf{x})\, \mathrm{d}s, \tag{11}$$

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.

Length: the distance traveled during a trajectory. We set $V(\mathbf{x}) = 1$ and use 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 $\mathrm{proj}.(\cdot)$ 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 $V(\mathbf{x}) = \min_{\mathbf{x}_o \in \mathcal{O}} ||\mathbf{x} - \mathbf{x}_o||$, where $\mathcal{O}$ is the obstacle space. We use the normalized integral.

Motion Legibility: how well the target can be predicted over the course of the exhibited trajectory. The exact definition is typically problem-specific, but if $V(\mathbf{x})$ measures the instantaneous legibility, we can use the normalized integral to measure the average legibility.

V. ROBOT CONTROLLERS

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) [25]. 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 $\mathbf{u}(\mathbf{x}) = \mathbf{x}^* - \mathbf{x}$, where $\mathbf{x}^*$ is the target in the task space, using the modulation matrix M proposed by [14]: $\mathbf{u}_M(\mathbf{x}) = M \cdot \mathbf{u}(\mathbf{x})$. M is calculated from a list of monotonically increasing Γ-functions, one per obstacle. We give a self-contained review of this approach in Supp. 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 1 presents the version that we use: first, a tree $\mathcal{T}$ is initialized with the start configuration s_0 as the root; then as long as the target configuration cannot be connected to the last node added to the tree (s_0 initially) with a straight line in the configuration space (C-space) using `attempt-grow`, a new point is sampled and, if possible, is connected to the nearest node in $\mathcal{T}$ through the same `attempt-grow`. The path from s_0 to s^* is returned after s^* is connected successfully. Unlike the DS, IL, and RL controllers, RRT is stochastic.

Algorithm 1: 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^*)

Note that 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 of RRT generally assures the algorithm will terminate in finite time with probability 1 if a path to the target exists [15]. 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. When the kernel is also uniform such that every g in the C-space is equally likely, this procedure reduces to using a sampled trajectory τ *as if* it were deterministic.

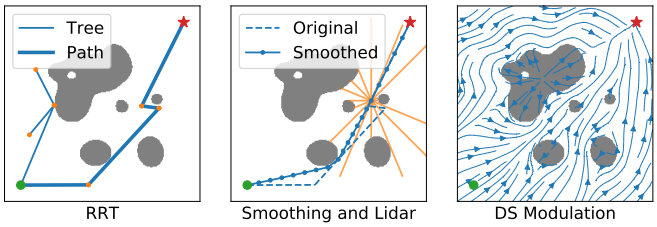

Fig. 3. RRT, IL and DS controllers on 2D navigation. Left: the RRT controller tree. Middle: smoothed RRT trajectory and lidar sensor (orange lines) for IL controller training. Right: the modulation by the DS controller.

VI. PROBLEM DOMAINS

We consider two common robotics tasks: 2D navigation of a point-mass robot, and 7DoF arm target reaching.

A. 2D Navigation

For the 2D navigation problem (Fig. 1, left), the environment is the area defined as $[x_{\text{min}}, x_{\text{max}}] \times [y_{\text{min}}, y_{\text{max}}]$. The goal is to navigate from $[x_{\text{start}}, y_{\text{start}}]$ to $[x_{\text{goal}}, y_{\text{goal}}]$. We define a flexible environment representation as a summation of radial basis function (RBF) kernels centered at so-called *obstacle points*. Specifically, given N_O obstacle points $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_{N_O} \in \mathbb{R}^2$, the environment is defined as

$$e(\mathbf{p}) = \sum_{i=1}^{N_O} \exp\left(-\gamma ||\mathbf{p} - \mathbf{p}_i||_2^2\right), \tag{12}$$

and each point $\mathbf{p}$ is an obstacle if $e(\mathbf{p}) > \eta$, for $\eta < 1$ to ensure each obstacle point $\mathbf{p}_i$ is exposed as an obstacle.

Our environments are bounded by $[-1.2, 1.2] \times [-1.2, 1.2]$, and the goal is to navigate from $[-1, -1]$ to $[1, 1]$. $N_O = 15$ and p_i coordinates are sampled uniformly in $x_i, y_i \in [-0.7, 0.7]$. A smaller γ and η makes the obstacles larger and more likely to be connected; we choose $\gamma = 25$ and $\eta = 0.9$. Supp. C shows random obstacle configurations demonstrating high diversity in this environment. We also implement a simple simulator: given the current robot position $[x, y]$ and the action $[\Delta x, \Delta y]$, the simulator clamps $\Delta x, \Delta y$ to the range of [-0.03, 0.03], and then moves the robot to $[x + \Delta x, y + \Delta y]$ if there is no collision, and otherwise simulates a frictionless inelastic collision (i.e. compliant sliding) that moves the robot tangent to the obstacle boundary.

We consider three controllers for this environment: an RRT planner, a deep learning IL policy, and a DS (Fig. 3).

The RRT planner implements Algorithm 1. After a path is found, we discretize the path to segments of length ≤ 0.03, and use each segment as $[\Delta x, \Delta y]$ in the simulator.

The IL controller uses smoothed RRT trajectories as expert demonstrations. It learns to predict heading angle from its current position and lidar readings in 16 equally spaced directions. More details can be found in Supp. D.

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. More details on the DS implementation are in Supp. E.

B. 7DoF Arm Reaching

For the arm reaching problem, we consider a 7DoF Franka Panda arm mounted on the side of a table. Starting from the same initial configuration on top of the table, the arm needs to reach a random location on either side under a T-shaped divider (Fig. 1 right). Since the initial configuration and the divider setup are fixed, a task is parameterized by the 3-dimensional target location. The simulation is done in PyBullet [26]. We again consider three controllers: an RRT planner, a deep RL PPO agent, and a DS formulation.

RRT implements Algorithm 1 and discretizes the path into segments to use in position control. Since the target is specified in the task space, inverse kinematics (IK) first finds the target joint configuration. More details are in Supp. F.

For RL, the state specifies the current robot joint configuration, the current end-effector location, and the target location. The action specifies the movement for each joint, clipped to 0.05 for position control. Both the actor and critic are implemented as standard multi-layer perceptron (MLP) networks with ReLU activation. The network architectures and other tranining details are in Supp. G.

DS outputs the end-effector trajectory in the task space, which is converted to joint space via IK. In order to successfully apply the modulation [14] in overly constrained environments, we implemented several adaptations including workspace bounding and a single Γ-function for multiple obstacles defined by a support vector machine (SVM) as in [27]. More details are in Supp. H.

VII. RESULTS

In this section, we present several insights from studying various behaviors on the controllers. For all MCMC sampling, we chose $\alpha = 0.1$ and used a truncated Gaussian transition kernel. Details are in Supp. I.

A. 2D Navigation

Straight-Line Deviation: For many obstacle configurations, the straight line path from the robot's initial position to the target position is not viable. We first sample obstacle configurations and trajectories that deviate minimally from this straight line path. Since the deviation is always non-negative, we use the match-type posterior in Eq. 3 with target $b^* = 0$. The top row of Fig. 4 plots posterior trajectories in orange, with prior trajectories in blue, for all three controllers. The bottom row plots the obstacle distributions with respect to the prior, with regions in red being more likely to be occupied by obstacles and those in blue less likely to be obstructed.

The visualization shows that for the DS and RRT controllers, the posterior trajectories and obstacle configurations are mostly symmetric with respect to the straight-line connection. This is expected as both methods are formulated

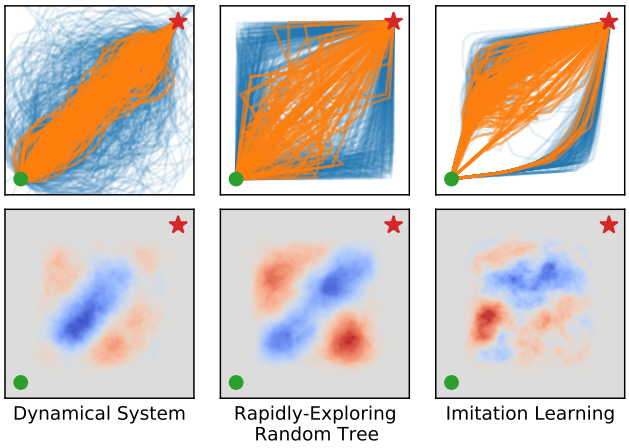

Fig. 4. For three 2D Navigation controllers, we sample a posterior that exhibits minimal straight-line deviation behavior, and see diverse outcomes. *Top:* Posterior trajectories in orange vs. prior in blue. *Bottom:* Posterior obstacle distribution, with red for higher obstacle density and blue for lower.

symmetrically with respect to the x- and y-coordinates. However, the symmetry is noticeably absent for IL: paths that deviate less from the straight-line connection are far more likely to favor the upper left direction. Since the neural architecture is symmetric with respect to coordinate system, we conclude that stochasticity in the dataset generation and training procedure leads to such imbalanced behaviors.

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. However, obstacle configurations for the DS and IL controllers appear counter-intuitive at first.

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 Supp. B). By taking advantage of anchor-like obstacles at the ends of the corridor, the modulation can reliably minimize the straight-line deviation.

For IL, looking at the trajectories and the obstacles together, we can see that the controller mostly takes the path upward, despite the heavy presence of obstacles in that direction. While this behavior is counterintuitive, if we reason about the dataset generation procedure, we can make sense of it: the smoothing procedure (Fig. 3 middle) results in most demonstrations navigating tightly around obstacles, and it is natural to expect that the trained IL controller will display the same behavior.

Legibility: We define the instantaneous legibility as the cosine similarity between the current robot direction and the direction to target $\mathbf{x}^*$, $V(\mathbf{x}) = \dot{\mathbf{x}} \cdot (\mathbf{x}^* - \mathbf{x}) / (||\dot{\mathbf{x}}|| \cdot ||\mathbf{x}^* - \mathbf{x}||)$, with the intuition that a particular run may be confusing to users if the robot does not align to the target most of the time. Even though this quantity is bounded by $[-1, 1]$, a general legibility definition may not be. Thus, we use the maximal mode of RoCUS to find DS trajectories and obstacle configurations that achieve *minimal* legibility, by negating $V(\mathbf{x})$ first. The left two panels of Fig. 5 present the samples. As expected, most trajectories take large detours due to the presence of obstacles in the center.

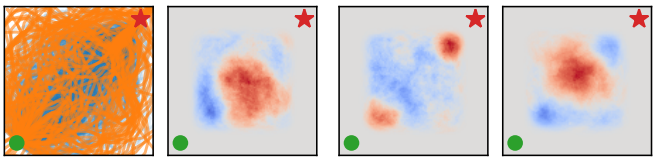

Fig. 5. Left 2: trajectories and obstacle configurations corresponding to sampling minimal DS legibility. Right 2: obstacle configurations for DS obstacle clearance behavior, minimizing and maximizing respectively.

Obstacle Clearance: We take $V(\mathbf{x}) = \min_{\mathbf{x}_o \in \mathcal{O}} ||\mathbf{x} - \mathbf{x}_o||$. For the DS, we sample two posteriors to maximize and minimize this behavior. As shown in the right two panels of Fig. 5, when minimizing obstacle clearance, we see clusters of obstacles in close proximity to the starting and target positions, such that the robot is forced to navigate around them. When maximizing obstacle clearance, we instead see central clusters of obstacles, such that the trajectories can avoid the obstacle by bearing hard left or right. Additional studies and quantitative summaries are in Supp. J.

B. 7DoF Arm Reaching

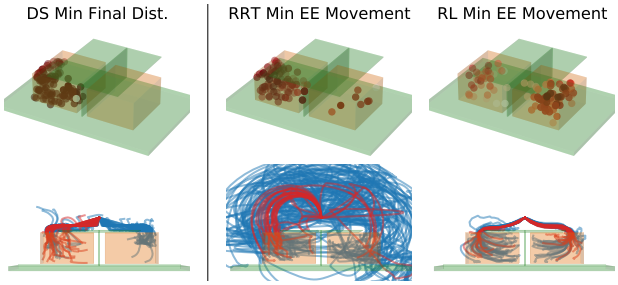

Fig. 6. Minimal final end-effector distance to target behavior for DS and minimal end-effector movement behavior for RRT and RL. Top: posterior targets locations, with tabletop + divider in green and target region in orange. Bottom: corresponding posterior trajectories in red, along with those under the prior in blue. The robot is mounted on the near long side of the table.

For this problem, RRT and RL are quite successful in reaching the target while the DS is not. For the latter, since the modulation is in the task space, the bulky robot structure and its close proximity to the divider result in a lot of collision when the end-effector is controlled by the DS. Hence, DS is evaluated on its minimal **final distance** behavior, defined as the distance from the end-effector to the target at the end of the execution, while RRT and RL are evaluated on their minimal **end-effector movement** behaviors, defined as total travel distance of the end-effector.

Fig. 6 (top) shows the posterior target locations for the three controllers, with darker colors indicating shorter behavior values. The target locations for both DS and RRT are highly asymmetric and strongly favor the left side, which is not the case for RL.

To understand the asymmetry, we compare the posterior trajectories to the prior for DS and RRT in Fig. 6 (bottom). Indeed, for DS, the end-effector rarely moves below the horizontal divider for right targets, but does succeed with a decent chance for left ones. Similarly for RRT, there are straight-line connections in the configuration space from the initial pose to some target regions on the left, while the entire right side requires at least an intermediate node. These two observations suggest that the left side is less "congested" with obstacles than the right side in the configuration space due to the asymmetric robot kinematic structure, which could be easily overlooked without such analysis and visualization. By comparison, the data-driven RL controller is able to learn efficient policies for both sides, despite such asymmetry.

Moreover, recognizing that DS almost always gets stuck when reaching to the right, we tuned its parameters so that the modulated end-effector path has larger obstacle clearance. Indeed, this greatly helps as shown in (Fig. 7), where posterior samples are quite balanced in two sides.

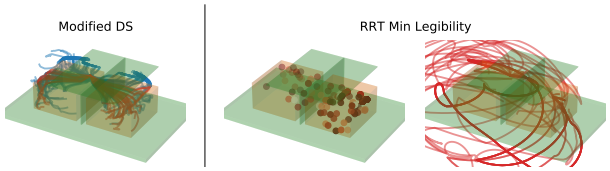

Fig. 7. Left: prior (blue) and posterior (red) samples for minimal final distance behavior for modified DS. Right: posterior samples for minimal legibility behavior for RRT. The robot is mounted on the near long side.

Legibility: We define legibility of reaching to the target on one side of the vertical divider as the average negative distance that the end effector moves in the other direction, $V(\mathbf{x}) = -\max(\tilde{\mathbf{x}}_1, 0)$, where $\tilde{\mathbf{x}}_1 = \mathbf{x}_1$ if target is on the left, or $\tilde{\mathbf{x}}_1 = -\mathbf{x}_1$ otherwise, and $\mathbf{x}_1$ is the x-coordinate of the robot end effector with right in the positive direction. We find target locations that are minimally legible and apply the maximal inference mode on the maximum distance measure.

We did not find any illegible motions from RL controllers for 2,000 targets, which is mostly expected since the RL reward is distance to the target. For RRT, however, since we are not using an optimal formulation (e.g. [28], [29]) or performing post-hoc smoothing, the controller is expected to frequently exhibit low legibility. Fig. 7 plots the posterior target locations and trajectories. Consistent with our findings above, the target locations leading to illegible motions are spread out mostly uniformly on the right, but concentrated in far-back area on the left. The trajectory plot confirms the illegibility. Additional results are in Supp. K.

VIII. CONCLUSION

Complementing existing task-completion-centric evaluations on hand-designed scenarios, ROCUS automatically generates scenarios and trajectories for specific target behaviors in a principled way to help humans build better mental models of the robots. In two common application domains, it gives non-obvious insights on implicit model biases due to mathematical formulation, model training or intrinsic robot structure, and helps debug and improve a controller.

An important step before actual deployment is to design appropriate user interfaces to facilitate the two-way communication between ROCUS and end-users—behavior specification and sample visualization, preferably with minimal or no programming required. Another direction is to develop downstream methods to further analyze the samples by drawing inspiration from the explainable artificial intelligence (XAI) community.

REFERENCES

[1] L. Sanneman, C. Fourie, and J. A. Shah, "The state of industrial robotics: Emerging technologies, challenges, and key research directions," *arXiv preprint arXiv:2010.14537*, 2020.

[2] J. Clark and D. Amodei, "Faulty reward functions in the wild," *URL https://blog. openai. com/faulty-reward-functions*, 2016.

[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*, vol. 36, no. 3, pp. 835–854, 2020.

[4] 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.

[5] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 63–78.

[6] S. Booth, Y. Zhou, A. Shah, and J. Shah, "Bayes-TrEx: Model transparency by example," *arXiv preprint arXiv:2002.10248*, 2020.

[7] B. Cohen, I. A. Şucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 589–595.

[8] 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*, vol. 1, no. open-issue, 2015.

[9] M. Moll, I. A. Sucan, and L. E. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.

[10] 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*, vol. 3, no. 4, pp. 3765–3772, 2018.

[11] 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," *arXiv preprint arXiv:1807.06757*, 2018.

[12] A. D. Dragan, K. C. Lee, and S. S. Srinivasa, "Legibility and predictability of robot motion," in *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2013.

[13] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[14] 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.

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

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

[17] 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*, 2019, pp. 220–229.

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

[19] 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*, vol. 568, no. 7753, pp. 477–486, 2019.

[20] 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," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 4, pp. 1440–1442, 2018.

[21] 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," *arXiv preprint arXiv:1906.08236*, 2019.

[22] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "RLBench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

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

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

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

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

[27] S. S. Mirrazavi Salehian, N. Figueroa, and A. Billard, "A unified framework for coordinated multi-arm motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1205–1232, 2018.

[28] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[29] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.

[30] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[31] S. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[32] Y. Zhou and K. Hauser, "6DOF grasp planning by optimizing a deep learning scoring function," in *Robotics: Science and Systems Workshop on Revisiting Contact – Turning a Problem into a Solution*, 2017.

[33] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.

[34] K. Hauser, "Robust contact generation for robot simulation with unstructured meshes," in *Robotics Research*. Springer, 2016.

SUPPLEMENTARY MATERIALS

A. Scale-Invariance and "Volume" Interpretation of α

We show that Eq. 4 results in the formulation being scale-invariant with respect to b. Consider the same behavior under two different units b_1 and b_2 with $b_1 = c \cdot b_2$. For example, b_1 can be the trajectory length in centimeters and b_2 is the same quantity but in meters, and $c = 100$. Thus, $p(c \cdot b_1) = p(b_2)$ and $b_1^* = c \cdot b_2^*$. To maintain the same α level in Eq. 4, we need to have $\sigma_1 = c \cdot \sigma_2$. This implies that

$$p(t, \tau | \hat{b}_1 = b_1^*) = \frac{\mathrm{No}(b_1^*; b(\tau, t), \sigma_1^2) p(\tau|t)\pi(t)}{p(\hat{b}_1 = b_1^*)} \quad (13)$$

$$= \frac{\mathrm{No}(b_2^*; b(\tau, t), \sigma_2^2) p(\tau|e)\pi(t)}{p(\hat{b}_2 = b_2^*)} = p(t|\hat{b}_2 = b_2^*) \quad (14)$$

because $\mathrm{No}(b_1^*; b(\tau, t), \sigma_1^2) = \mathrm{No}(b_2^*; b(\tau, t), \sigma_2^2)$ due to the same scaling of $b_1 \sim b_2$ and $\sigma_1 \sim \sigma_2$, and $p(\hat{b}_1 = b_1^*) = p(\hat{b}_2 = b_2^*)$ as they are the same event. We conclude the posterior distribution is scale-invariant with respect to $b(\tau, t)$.

To motivate the bound of $[b^* - \sqrt{3}\sigma, b^* + \sqrt{3}\sigma]$ in Eq. 4, we consider a uniform approximation to $\mathrm{No}(b^*, \sigma^2)$. To match the mean b^* and standard deviation σ, $\mathrm{Unif}(b^* - \sqrt{3}\sigma, b^* + \sqrt{3}\sigma)$ is needed. If we use this uniform distribution in Eq. 2, the posterior Eq. 3 can be instantiated by sampling from the prior and rejecting tasks for which the trajectory behavior $b(\tau, t)$ falls outside of this bound. Thus, Eq. 4 specifies that the "volume" of $(\alpha \cdot 100)\%$ under $p(t, \tau)$ is maintained.

The same invariance and "volume" interpretation holds for Eq. 8 as well. The former stems from the standardization on b performed in Eq. 6. The latter uses the same uniform approximation but the bound is one-sided since $\beta \in (0, 1)$ by nature of the sigmoid transformation.

B. Dynamical System Modulation

We review the DS formulation proposed by [14], and present our problem-specific adaptations for 2D Navigation (Supp. E) and 7DoF arm reaching (Supp. H). A reader familiar with DS motion controllers may skip this review.

Given a target $\mathbf{x}^*$ and the robot's current state $\mathbf{x}$, a linear controller $\mathbf{u}(x) = \mathbf{x}^* - \mathbf{x}$ will guarantee convergence of $\mathbf{x}$ to $\mathbf{x}^*$ if there are no obstacles. However, it can easily get stuck in the presence of obstacles. [14] proposes a method to calculate a modulation matrix $M(\mathbf{x})$ at every $\mathbf{x}$ such that if the new controller follows $\mathbf{u}_M(\mathbf{x}) = M(\mathbf{x}) \cdot \mathbf{u}(\mathbf{x})$, then $\mathbf{x}$ still converges to $\mathbf{x}^*$ but never gets stuck, as long as $\mathbf{x}^*$ is in free space. In short, the objective of the DS modulation is to preserve the linear controller's convergence guarantee while also ensuring that the robot is never in collision.

The modulation matrix $M(\mathbf{x})$ is computed from a list of obstacles, each of which is represented by a Γ-function. For the i-th obstacle $\mathcal{O}_i$, its associated gamma function Γ_i must satisfy the following properties:

- $\Gamma_i(\mathbf{x}) \leq 1 \iff \mathbf{x} \in \mathcal{O}_i$,
- $\Gamma_i(\mathbf{x}) = 1 \iff \mathbf{x} \in \partial\mathcal{O}_i$,
- $\exists\, \mathbf{r}_i, \text{s.t.} \forall\, t_1 \geq t_2 \geq 0, \forall\, \mathbf{u}, \Gamma_i(\mathbf{r}_i + t_1\mathbf{u}) \geq \Gamma_i(\mathbf{r}_i + t_2\mathbf{u})$.

In words, the Γ-function value needs to be less than 1 when inside the obstacle, equal to 1 on the boundary, greater than 1 when outside. This function must also be monotonically increasing radially outward from a specific point $\mathbf{r}_i$. This point is dubbed the *reference point*. From this formulation, $\mathbf{r}_i \in \mathcal{O}_i$ and any ray from $\mathbf{r}_i$ intersects with the obstacle boundary $\partial\mathcal{O}_i$ exactly once. The latter property is also the definition that $\mathcal{O}_i$ is "star-shaped" (Fig. 11). For most common (2D) geometric shape such as rectangles, circles, ellipses, regular polygons and regular star polygons, $\mathbf{r}_i$ can be chosen as the geometric center.

We first consider the case of a single obstacle $\mathcal{O}$, represented by Γ with reference point $\mathbf{r}$. Use d to denote the dimension of the space. We define

$$M(\mathbf{x}) = E(\mathbf{x})D(\mathbf{x})E^{-1}(\mathbf{x}). \quad (15)$$

We have

$$E(x) = [\mathbf{s}(\mathbf{x}), \mathbf{e}_1(\mathbf{x}), ..., \mathbf{e}_{d-1}(\mathbf{x})], \quad (16)$$

where

$$\mathbf{s}(\mathbf{x}) = \frac{\mathbf{x} - \mathbf{r}}{||\mathbf{x} - \mathbf{r}||} \quad (17)$$

is the unit vector in the direction of $\mathbf{x}$ from $\mathbf{r}$, and $\mathbf{e}_1(\mathbf{x}), ...,$ $\mathbf{e}_{d-1}(\mathbf{x})$ form a $d-1$ orthonormal basis to the gradient of the Γ-function, $\nabla\Gamma(\mathbf{x})$ representing the normal to the obstacle surface. $D(\mathbf{x})$ is a diagonal matrix whose diagonal entries are $\lambda_s, \lambda_1, ..., \lambda_{d-1}$, with

$$\lambda_s = 1 - \frac{1}{\Gamma(\mathbf{x})}, \quad (18)$$

$$\lambda_1, ..., \lambda_{d-1} = 1 + \frac{1}{\Gamma(\mathbf{x})}. \quad (19)$$

each eigenvalue determines the scaling of each direction. Conceptually, as the robot approaches the obstacle, this modulation decreases the velocity for the component in the reference point direction (i.e. toward obstacles) while increases velocity for perpendicular components. The combined effect results in the robot being deflected away tangent to the obstacle surface.

With N obstacles, we compute the modulation matrix $M_i(\mathbf{x})$ for every obstacle using the procedure above and the individual controllers $\mathbf{u}_{M_i}(\mathbf{x}) = M_i(\mathbf{x}) \cdot \mathbf{u}(\mathbf{x})$. The final modulation is the aggregate of all the individual modulations. However, a simple average is insufficient since closer obstacles should have higher influence to prevent collisions.

[14] proposed the following aggregation procedure. Let $\mathbf{u}_i$ denote the individual modulations, and $\mathbf{u}$ denote the final aggregate modulation. Let n_i to denote the norm of $\mathbf{u}_i$. Defines $\mathbf{u}$ to be

$$\mathbf{u} = n_a \mathbf{u}_a, \quad (20)$$

where n_a is the aggregate norm, and $\mathbf{u}_a$ is the aggregate direction.

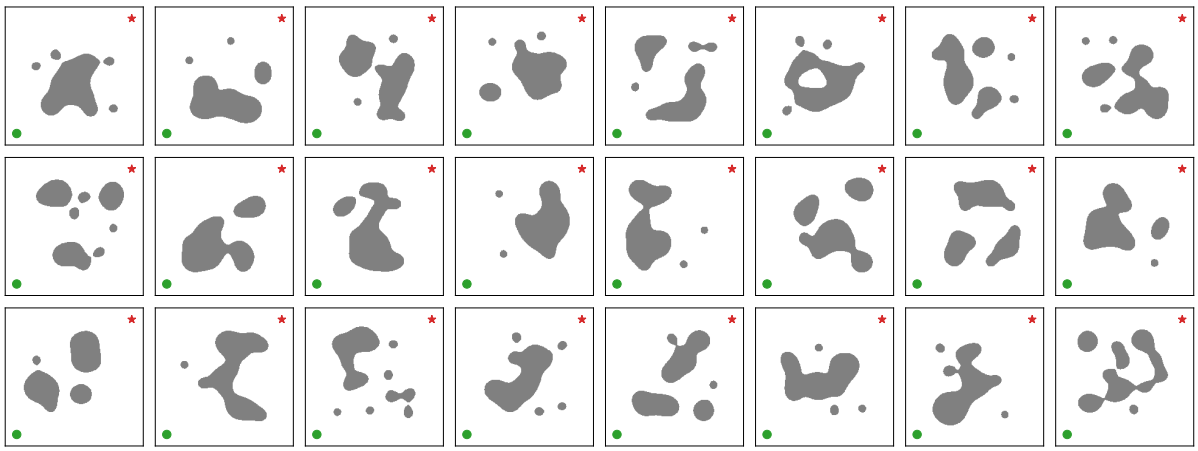

Fig. 8. An assortment of randomly generated RBF 2D environments, providing a sense of the diversity generated with this formulation. The green dots are the environment starting points and the red stars are navigation targets. We show DS modulation examples for the first three environments in Fig. 10.

The aggregate norm is computed as

$$n_a = \sum_{i=1}^{N} w_i n_i, \tag{21}$$

$$w_i = \frac{b_i}{\sum_{j=1}^{N} b_j}, \tag{22}$$

$$b_i = \prod_{1 \le j \le N, j \neq i} \Gamma_j(\mathbf{x}). \tag{23}$$

The above definition ensures that $\sum_{i=1}^{N} w_i = 1$, and $w_i \to 1$ when $\mathbf{x}$ approaches $\mathcal{O}_i$ (and only $\mathcal{O}_i$, which holds as long as obstacles are disjoint).

$\mathbf{u}_a$ is instead computed using what [14] calls "$\boldsymbol{\kappa}$-space interpolation." First, similar to the basis vector matrix $E(\mathbf{x})$ introduced above, we construct another such matrix, but with respect to the original controller $\mathbf{x}^* - \mathbf{x}$. We denote it as $R = [(\mathbf{x}^* - \mathbf{x})/||\mathbf{x}^* - \mathbf{x}||, \mathbf{e}_1, ..., \mathbf{e}_{d-1}]$, where $\mathbf{e}_1, ..., \mathbf{e}_{d-1}$ are again orthonomal vectors spanning the null space.

For each $\mathbf{u}_i$, we compute its coordinate in this new R-frame as $\hat{\mathbf{u}}_i = R^{-1}\mathbf{u}_i$. Its $\boldsymbol{\kappa}$-space representation is

$$\boldsymbol{\kappa}_i = \frac{\arccos(\hat{\mathbf{u}}_i^{(1)})}{\sum_{m=2}^{d} \hat{\mathbf{u}}_i^{(m)}} \left[\hat{\mathbf{u}}_i^{(2)}, ..., \hat{\mathbf{u}}_i^{(d)}\right]^T \in \mathbb{R}^{d-1}, \tag{24}$$

where the superscript (m) refers to the m-th entry. $\boldsymbol{\kappa}_i$ is a scaled version of the $\hat{\mathbf{u}}_i$ with the first entry removed. We perform the aggregation in this $\boldsymbol{\kappa}$-space using the weights w_i calculated above (25), transform it back to the R-frame (26), and finally transform it back to the original frame (27):

$$\boldsymbol{\kappa}_a = \sum_{i=1}^{N} w_i \boldsymbol{\kappa}_i \tag{25}$$

$$\hat{\mathbf{u}}_a = \left[\cos(||\boldsymbol{\kappa}_a||), \frac{\sin(||\boldsymbol{\kappa}_a||)}{||\boldsymbol{\kappa}_a||}\boldsymbol{\kappa}_a^T\right]^T \tag{26}$$

$$\mathbf{u}_a = R\hat{\mathbf{u}}_a. \tag{27}$$

As mentioned in Eq. 20, the final modulation is $\mathbf{u} = n_a \mathbf{u}_a$.

1) Tail-Effect: An artifact of the above formulation is the "tail-effect," where the robot is modulated to go around the obstacle even when it has passed by the obstacle and the remaining trajectory has no chance of collision under the non-modulated controller. This effect has been observed in [30] for a related but different type of modulation. Fig. 9 is reproduced from [30] (Fig. 7) showing the tail effect on the left and its removal on the right.

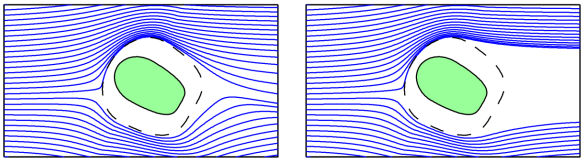

Fig. 9. Tail effect (left) and its removal (right), reproduced from Fig. 7 in [31]. The target is on the far right side.

This tail effect induces the placement of obstacles at the end of the "diagonal corridor" as seen in our straight-line deviation experiments (Fig. 4, left). We can modify the modulation to remove this effect.

C. RBF-Defined Environment Visualization

Fig. 8 depicts a randomly selected assortment of 2D environments. These environments demonstrate the flexibility and diversity of the RBF environment definition.

D. IL Controller for 2D Navigation

The imitation learning controller is a memoryless policy implemented as a fully connected neural network with two hidden layers of 200 neurons each and ReLU activations. The input is 18 dimensional, with two dimensions for the current (x, y) position of the robot, and 16 dimensions for a lidar sensor in 16 equally-spaced directions, with a maximum range of 1. The network predicts the heading angle θ, and the controller operates on the action of $[\Delta x, \Delta y] = [0.03\cos\theta, 0.03\sin\theta]$.

The network is trained on smoothed RRT trajectories. Specifically, we use the RRT controller to find and discretize

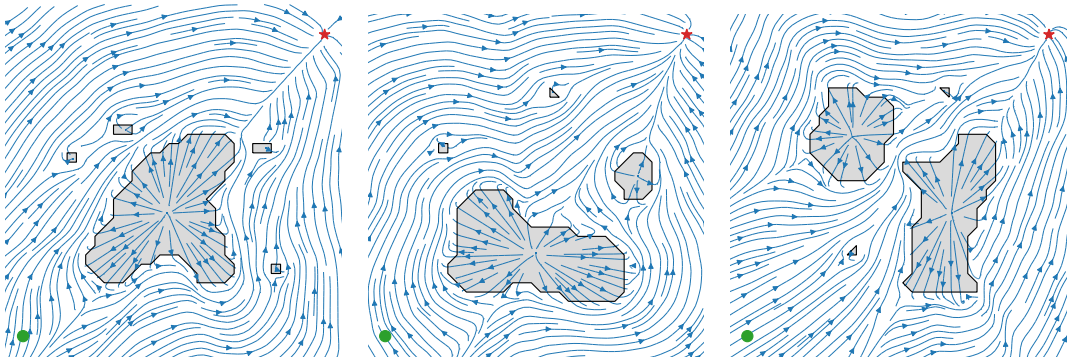

Fig. 10. Streamlines showing the modulation effect of the dynamical system for three 2D navigation tasks. The environments correspond to the first three examples of Fig. 8. Green dots are starting positions and red stars are navigation targets.

a trajectory. Then the smoothing procedure repeatedly replaces each point by the mid-point of its two neighbors, absent collisions. When this process converges, each point on the trajectory becomes one training data point.

Since only local observations are available and the policy is memoryless, the robot may get stuck in obstacles, which happens in approximately 10% of the runs. In addition, while the output target is continuous, a regression formulation with mean-squared error (MSE) loss is inappropriate, due to multimodality of the output. For example, when the robot is facing an obstacle, moving to either left or right would avoid it, but if both directions appear in the dataset, the MSE loss would drive the prediction to be the average, resulting in a head-on collision. This problem has been recognized in other robotic scenarios such as grasping [32] and autonomous driving [33]. We follow the latter to treat this problem as classification with 100 bins in the $[0, 2\pi]$ range.

E. DS Controller for 2D Navigation

For the DS controller, there are two technical challenges in using [14] on our RBF-defined environment. First, we need to identify and isolate each individual obstacle, and second, we need to define a Γ-function for each obstacle.

To find all obstacles, we discretize the environment into an occupancy grid of resolution 150×150 covering the area of $[-1.2, 1.2] \times [-1.2, 1.2]$. Then we find connected components using flood fill, and each connected component is taken to be an obstacle.

To define a Γ-function for each obstacle, we first choose the reference point as the center of mass of the connected component. Then we cast 50 rays in 50 equally spaced directions from the reference point and find the intersection point of each ray with the boundary of the connected component. Finally, we connected those intersections in sequence and get a polygon. In case of multiple intersection points, we take the farthest point as vertex of the polygon, essentially completing the non-star-shaped obstacle to be star-shaped, as shown in Fig. 11.

Given an arbitrary point $\mathbf{x}$, we define

$$\Gamma(\mathbf{x}) = \frac{||\mathbf{x} - \mathbf{r}||}{||\mathbf{i} - \mathbf{r}||}, \tag{28}$$

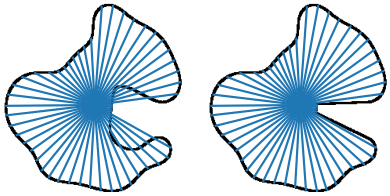

Fig. 11. Left: an obstacle which is not star-shaped. Some radial lines extending from the obstacle's reference point cross the boundary of the obstacle twice. Right: the same obstacle, modified to instead be star-shaped.

where $\mathbf{r}$ is the reference point and $\mathbf{i}$ is the intersection point with the polygon of the ray from $\mathbf{r}$ in $\mathbf{x}-\mathbf{r}$ direction. It is easy to see that this Γ definition satisfies all three requirements for Γ-functions listed in Supp. B.

Finally, to compensate for numerical errors in the process (e.g. approximating obstacles with polygons), we define the control inside obstacle to be the outward direction, which helps preventing the robot from getting stuck at obstacle boundaries in practice. Three examples of DS modulation of the 2D navigation environment are shown in Figure 10.

F. RRT Controller for 7DoF Arm Reaching

Since the target location is specified in the task space, we first find the target joint space configuration using inverse kinematics (IK). The initial configuration starts with the arm positioned down on the same side as the target. If the IK solution is in collision, we simulate the arm moving to it using position control, and redefine the final configuration at equilibrium as the target (i.e. its best effort reaching configuration). We solve the IK using Klamp't [34].

G. RL Controller for 7DoF Arm Reaching

The RL controller implements the proximal policy gradient (PPO) algorithm [25]. The state space is 22-dimensional and consists of the following:

- 7D joint configuration of the robot,
- 3D position of the end-effector,
- 3D roll-pitch-yaw of the end effector,
- 3D velocity of the end-effector,
- 3D position of the target,
- 3D relative position from the end-effector to the target.

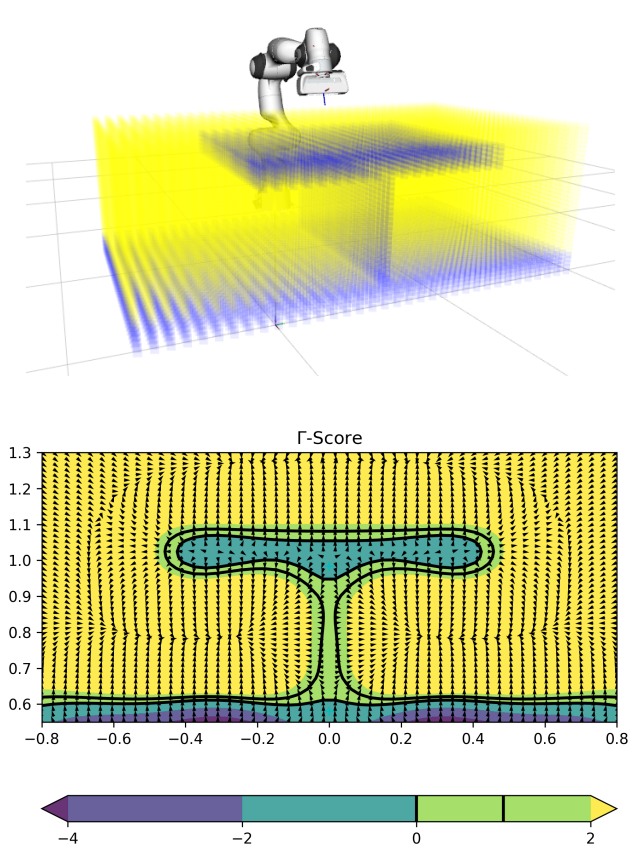

Fig. 12. Above: the division of 3D space as either containing an obstacle or free space. This data is used to train an SVM, which acts as an interpolator. The classification scores of the SVM are used as the Γ function for this 3D reaching task. Below: a 2D slice showing the smoothed Γ scores.

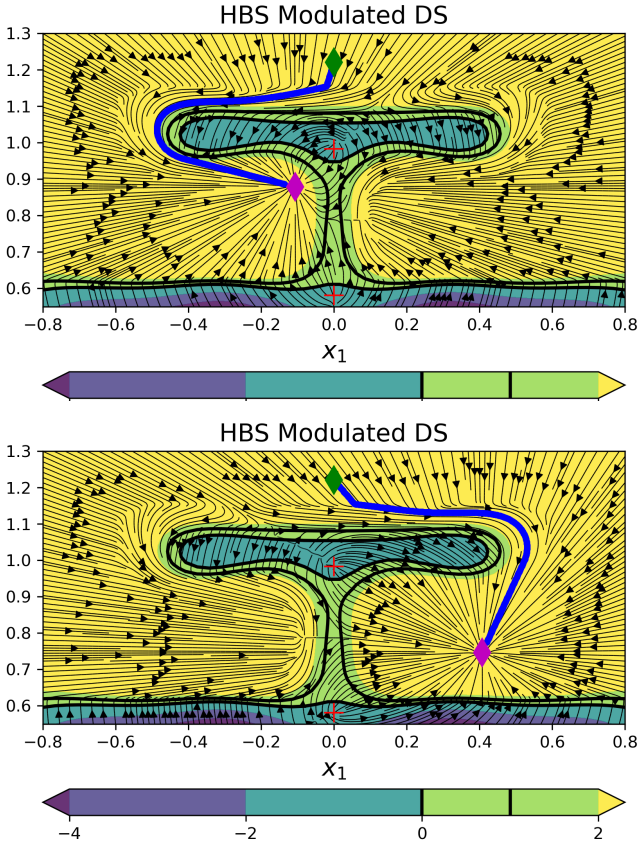

Fig. 13. Cross-sections showing streamlines of the dynamical system modulation effect for two distinct targets in the 3D reaching task. Red crosses indicate reference points. Green diamond is the initial position of the end-effector for all experiments.

The action is 7-dimensional for movement in each joint, which is capped at $[-0.05, 0.05]$.

Both the actor and the critic are implemented with fully connected networks with two hidden layers of 200 neurons each, and ReLU activations. The action is parametrized as Gaussian where the actor network predicts the mean, and 7 standalone parameters learns the log variance for each of the 7 action dimensions. At test time, the policy deterministically outputs the mean action given a state.

H. DS Controller for 7DoF Arm Reaching

For the DS controller in the 7DoF arm reaching task, we face the same challenges as in the 2D navigation task. Namely, defining an appropriate Γ-function for the obstacle configuration that holds the three properties introduced in [14] (listed in Supp. B). Additionally, as the DS modulation technique does not consider the robot's morphology, end-effector shape or workspace limits (as it only modulates a point-mass) we implement several adaptations to allow for this technique to be used properly with a 7DoF robot arm. Specifically, the modulated linear controller $\mathbf{u}_M(\mathbf{x}) = M(\mathbf{x}) \cdot \mathbf{u}(\mathbf{x})$, with a modulation matrix $M(\mathbf{x})$ computed as in [14] and described in Supp. B, controls for the 3D position of the tip of the end-effector. This 3D position is shown as the small reference frame at the edge of the rectangular surface of the end-effector in Fig. 12. The desired velocity of the end-effector tip, given by the modulated linear controller, is then tracked by the 7DoF arm via the position-level IK solver used in the RRT implementation described in Supp. F. Following we describe the details of these adaptations.

1) SVM-Defined Γ-function for Constrained Workspaces: The original approach for the DS modulation strategy [30] uses spherical or ellipsoidal geometric representations to define the Γ-functions pertaining to each obstacle. Such representations are also used in the improved approach that we adopt in this work [14]. In such works, when multiple convex obstacles are present in the environment, these are defined by ellipsoids that encapsulate the entire obstacle. On the other hand, if the obstacles are non-convex and star-shaped, a collection of ellipsoids with a common intersection is used to parametrize the Γ-functions. This approach works well in practice, however, representing star-shaped objects as collections of ellipsoids or even convex objects as ellipsoids might lead to undesirable behaviors. They can produce either overly conservative collision avoidance behaviors or an under-specified Γ-function that cuts corners and wrongfully leads to collisions at the edges of tables or other type of rectangular-shaped objects. Hence, the success of this DS modulation strategy relies heavily on the correct placement and fitting of the ellipsoids on the obstacles and also the

correct placement of the reference points $\mathbf{r}_i$ corresponding to the i-th obstacle $\mathcal{O}_i$. In the 2D navigation task, we solved this issue by representing the obstacles as polygonal-based Γ-functions (see Supp. E). This approach works well in 2D for disconnected obstacles, yet, it is not easily transferable to 3D. Furthermore, in the case of our proposed constrained table-top environment the obstacle configuration is not star-shaped, hence, a more general Γ-function is necessary.

To avoid the geometric error-prone Γ-functions, we take inspiration from prior work in data-driven self-collision avoidance boundary learning [27]. In this work, a single Γ-function is used to represent the free-space and collided regions of the robot's workspace in joint-space coordinates. Given a dataset of collided and non-collided robot configurations, a Γ-function of class $\mathcal{C}^1$ is learned by framing the problem as a binary classification solved via support vector machines (SVM). As shown in Fig. 12, we discretize the 3D workspace of the robot and generate a dataset of collided positions as (-1) class and the free-space as (+1) class. By using the radial basis function (RBF) kernel $K(\mathbf{x}_1, \mathbf{x}_2) = e^{-\gamma||\mathbf{x}_1 - \mathbf{x}_2||^2}$, where parameter γ defines kernel width, the SVM decision function $\Gamma(\mathbf{x})$ has the following form:

$$\begin{aligned}\Gamma(\mathbf{x}) &= \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b \\ &= \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\mathbf{x}-\mathbf{x}_i||^2} + b,\end{aligned} \tag{29}$$

and the equation for $\nabla\Gamma(\mathbf{x})$ is naturally derived as follows:

$$\begin{aligned}\nabla\Gamma(\mathbf{x}) &= \sum_{i=1}^{N_{sv}} \alpha_i y_i \frac{\partial K(\mathbf{x}, \mathbf{x}_i)}{\partial \mathbf{x}} \\ &= -\gamma \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\mathbf{x}-\mathbf{x}_i||^2} (\mathbf{x} - \mathbf{x}_i).\end{aligned} \tag{30}$$

In Eq. 29 and 30, $\mathbf{x}_i$ $(i = 1, ..., N_{sv})$ are the support vectors from the training dataset, y_i are corresponding collision labels (-1 if position is collided, $+1$ otherwise), $0 \leq \alpha_i \leq C$ are the weights for support vectors and $b \in \mathbb{R}$ is decision rule bias. Parameter $C \in \mathbb{R}$ is a penalty factor used to trade-off between errors minimization and margin maximization. We empirically set the hyper-parameters of the SVM to $C = 20$ and $\gamma = 20$. Parameters α_i and b and the support vectors $\mathbf{x}_i$ are estimated by solving the optimization problem for the soft-margin kernel SVM using the Python scikit-learn library.

2) 7DoF Arm Position Control with 3D Modulated DS: Given a desired modulated 3D velocity for the end-effector tip, $\dot{\mathbf{x}}_M = \mathbf{u}_M(\mathbf{x})$, we compute the next desired 3D position by numerical integration:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{u}_M(\mathbf{x}_t)\Delta t \tag{31}$$

where $\mathbf{x}_t, \mathbf{x}_{t+1} \in \mathbb{R}^3$ are the current and next desired 3D position of the tip of the end-effector and $\Delta t = 0.03$ is the control loop time step. $\mathbf{x}_{t+1}$ is then the target in Cartesian world space coordinates that defines the objective of the position-based IK solver implemented in Klamp't [34].

I. Transition Kernel in MCMC Sampling

We used a truncated Gaussian transition kernel for all experiments. For the RBF-defined 2D environment, we initialize 15 obstacle points with coordinates sampled uniformly in $[-0.7, 0.7]$. The transition kernel operates independently on each obstacle coordinate: given the current value of x, the kernel samples a proposal from $\mathrm{No}(\mu = x, \sigma^2 = 0.1^2)$ truncated to $[-0.7, 0.7]$ (and also appropriately scaled). For the arm reaching task, the target is sampled uniformly from two disjoint boxes, with the left box at $[-0.5, -0.05] \times [-0.3, 0.2] \times [0.65, 1.0]$ and the right box at $[0.05, 0.5] \times [-0.3, 0.2] \times [0.65, 1.0]$. Again, we use the same transition kernel with $\sigma_x = 0.1, \sigma_y = 0.03, \sigma_z = 0.035$ in three directions. Again, the distribution is truncated to the valid target region ($x \in [-0.5, -0.05] \cup [0.05, 0.5], y \in [-0.3, 0.2], z \in [0.65, 1.0]$). In other words, the transition kernel implicitly allows for the jump across two box regions.

In addition, the stochastic RRT controller also requires a transition kernel. As discussed in Sec. V, we initialize its values on an as-needed basis. When necessary, we sample a configuration uniformly between the lower- and upper-limit (i.e. $[x_L, x_U)$. For each configuration, the same Gaussian kernel truncated to $[x_L, x_U]$, with $\sigma = 0.1(x_U - x_L)$.

J. Additional Results for 2D Navigation

TABLE I

MEANS REPORTED FOR 500 SAMPLES DRAWN FROM EACH OF THE PRIOR AND POSTERIOR FOR EACH 2D NAVIGATION CONTROLLER.

Control	Metric	Target	Prior Mean	Posterior Mean
DS	Length	0	203	166
	Avg. Jerk	0	1.84e−3	1.46e−3
	Straight	0	0.256	0.084
	Legibility	min	0.819	0.650
	Obstacle	0	0.309	0.229
	Obstacle	max	0.309	0.611
IL	Length	0	161	161
	Avg. Jerk	0	6.95e−4	3.19e−4
	Straight	0	0.378	0.301
	Legibility	min	0.877	0.784
	Obstacle	0	0.262	0.218
	Obstacle	max	0.262	0.387
RRT	Length	0	182	174
	Avg. Jerk	0	4.24e−4	2.79e−4
	Straight	0	0.470	0.162
	Legibility	min	0.798	0.669
	Obstacle	0	0.312	0.241
	Obstacle	max	0.312	0.442

Table I presents a quantitative overview of the performance of our sampling procedure, allowing us to compare behaviors between different controllers. For example, we find the DS controller has the highest variability on the obstacle clearance min and max tasks: it has the largest obstacle avoidance metric mean (0.611), and the second lowest obstacle clearance metric mean $(0, 229)$. Where behavior metrics are positive real numbers (e.g., trajectory length, obstacle clearance), a

target of 0 is functionally equivalent to using the maximal mode approach (with $b^* \to -\inf$) target.

In addition to this full summary of the quantitative results, we include visual overviews of the maximum obstacle clearance behavior and the minimum obstacle clearance behaviors for all three controllers in Figure 14. For the DS formulation, obstacle avoidance is best achieved when the environment clusters obstacles in the center. In contrast, for both RRT and IL, obstacle avoidance is best achieved when the obstacles are clustered in the upper left corner. For minimal obstacle clearance behaviors, obstacles are clustered near the start and end positions for all three controllers.

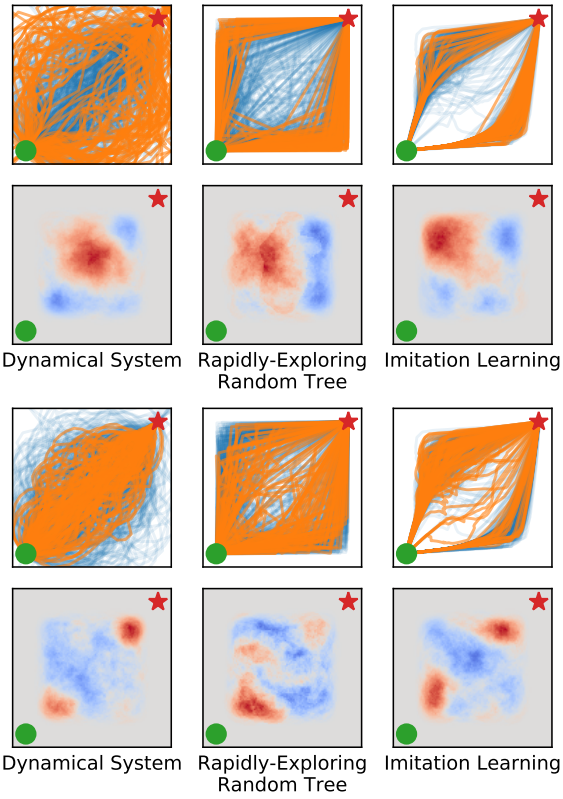

Fig. 14. First two rows: trajectories which maximize obstacle clearance from the prior (blue) and posterior (orange) for each controller, and the corresponding obstacle density plots (red indicates obstacles are more likely present). Below: the same, but for minimized obstacle clearance instead.

K. Additional Results for 7DoF Arm Reaching

Table II presents quantitative overviews of some behaviors for the 7DoF arm reaching task.

TABLE II

MEANS REPORTED FOR 500 SAMPLES DRAWN FROM EACH OF THE PRIOR AND POSTERIOR FOR EACH 7DOF ARM REACHER CONTROLLER.

Control	Metric	Target	Prior Mean	Posterior Mean
DS	Avg. Jerk	0	$2.31\mathrm{e}-4$	$5.98\mathrm{e}-5$
	Straight	0	0.980	0.913
	EE Dist	0	0.934	0.623
RL	Avg. Jerk	0	7.17e−3	5.50e−3
	Straight	0	0.858	0.762
	EE Dist	0	0.958	0.691
RRT	Avg. Jerk	0	1.87e−3	4.92e−4
	Straight	0	1.223	0.897
	EE Dist	0	3.741	1.192