

A Scale-Invariance and “Volume” Interpretation of α

We show that Eq. 3 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. 3, we need to have $\sigma_1 = c \cdot \sigma_2$. This implies that

$$p(t, \tau | \hat{b}_1 = b_1^*) = \frac{\mathcal{N}(b_1^*; b(\tau, t), \sigma_1^2) p(\tau | t) \pi(t)}{p(\hat{b}_1 = b_1^*)} \quad (6)$$

$$= \frac{\mathcal{N}(b_2^*; b(\tau, t), \sigma_2^2) p(\tau | t) \pi(t)}{p(\hat{b}_2 = b_2^*)} = p(t | \hat{b}_2 = b_2^*) \quad (7)$$

because $\mathcal{N}(b_1^*; b(\tau, t), \sigma_1^2) = \mathcal{N}(b_2^*; b(\tau, t), \sigma_2^2)$ 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 that 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. 3, we consider a uniform approximation to $\mathcal{N}(b^*, \sigma^2)$. To match the mean b^* and standard deviation σ , $\mathcal{U}(b^* - \sqrt{3}\sigma, b^* + \sqrt{3}\sigma)$ is needed. If we use this uniform distribution in Eq. 2 in lieu of the normal distribution, the posterior 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. 3 specifies that the “volume” of $(\alpha \cdot 100)\%$ under $p(t, \tau)$ is maintained.

The same invariance and “volume” interpretation holds for Eq. 5 as well. The former stems from the standardization on b performed in Eq. 4. 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 MCMC Sampling with Stochastic Dynamics

Using the same logic as the case of stochastic controller, ROCUS 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 case of stochastic controllers. In the real world, we can

- treat a sampled trajectory as the deterministic one;
- restart multiple times to estimate $\mathbb{E}_\tau[b(\tau, t)]$; or
- use likelihood-free MCMC methods [34].

We leave these investigations to future work, and use deterministic dynamics in our experiments.

C Mathematical Definitions of Behaviors

A versatile and general form of a behavior is the (normalized or unnormalized) line integral of some scalar field along the trajectory. Specifically, we have

$$b = \int_\tau V(\mathbf{x}) ds \quad \text{or} \quad b = \frac{1}{\|\tau\|} \int_\tau V(\mathbf{x}) ds. \quad (8)$$

Using this general definition, we define a list of behaviors in Tab. 2.

Trajectory length simply measures how long the trajectory is. In most of the behaviors below, the normalizing factor is also length to decorrelate the behavior value from it.

Average velocity, acceleration and jerk are useful for a general understanding about how fast and abruptly the robot moves, which is an important factor to its safety.

Straight-line deviation measures how much the robot trajectory deviates from the straight-line path, in either the task space or the state space. A specific task instance in which the straight-line path is feasible (e.g. with no obstacles) is typically considered easy. Thus, we can find tasks of varying difficulty level on the spectrum of deviation values. In the definition, \mathbf{x}_i is the initial state, \mathbf{x}_f is the final state, and proj is the projection operator.

Name	Definition	Name	Definition
Trajectory Length	$b = \int_{\tau} 1 ds$	Straight-Line Deviation	$b = \frac{1}{\ \tau\ } \int_{\tau} \ \mathbf{x} - \text{proj}_{\mathbf{x}_f - \mathbf{x}_i} \mathbf{x}\ ds$
Average Velocity	$b = \frac{1}{\ \tau\ } \int_{\tau} \ \dot{\mathbf{x}}\ ds$	Obstacle Clearance	$b = \frac{1}{\ \tau\ } \int_{\tau} \min_{\mathbf{x}_o \in \mathcal{O}} \ \mathbf{x} - \mathbf{x}_o\ ds$
Average Acceleration	$b = \frac{1}{\ \tau\ } \int_{\tau} \ \ddot{\mathbf{x}}\ ds$	Near-Obstacle Velocity	$b = \frac{\int_{\tau} \ \dot{\mathbf{x}}\ / \min_{\mathbf{x}_o \in \mathcal{O}} \ \mathbf{x} - \mathbf{x}_o\ ds}{\int_{\tau} 1 / \min_{\mathbf{x}_o \in \mathcal{O}} \ \mathbf{x} - \mathbf{x}_o\ ds}$
Average Jerk	$b = \frac{1}{\ \tau\ } \int_{\tau} \ \ddot{\mathbf{x}}\ ds$	Motion Legibility	$b = \frac{1}{\ \tau\ } \int_{\tau} p(g \mathbf{x}) ds$

Table 2: A list of behavior definitions.

Obstacle clearance measures the average distance to the closest obstacle. Finding situations in which the robot moves very close to obstacles is crucial to understanding the collision risk level. In the definition, \mathcal{O} represents the obstacle space.

Near-obstacle velocity calculates how fast the robot moves around obstacles. We define it as the average velocity on the trajectory weighted by the inverse distance to the closest obstacle. Other weighting method can be used, as long as it is non-negative and monotonically decreasing with distance. This behavior is correlated with the damage of a potential collision, as high-speed collisions are usually far more dangerous and costly. Since we want the value to represent the average velocity, we normalize by the integral of weights along the trajectory.

Motion legibility measures how well the goal can be predicted over the course of the exhibited trajectory. In our definition, we use $p(g|\mathbf{x})$, or the conditional probability of the goal g given at the current robot state \mathbf{x} , but there may be better application-specific definitions.

D Dynamical System Modulation

We review the DS formulation proposed by Huber et al. [7], and present our problem-specific adaptations for 2D Navigation in App. H.2 and 7DoF arm reaching in App. J.3. 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 $\mathbf{u}(\mathbf{x}) = \mathbf{x}^* - \mathbf{x}$ 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. Huber et al. [7] 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. 12). For most common (2D) geometric shape such as rectangles, circles, ellipses, regular polygons and regular stars, \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

$$M(\mathbf{x}) = E(\mathbf{x})D(\mathbf{x})E^{-1}(\mathbf{x}). \quad (9)$$

We have

$$E(x) = [\mathbf{s}(\mathbf{x}), \mathbf{e}_1(\mathbf{x}), \dots, \mathbf{e}_{d-1}(\mathbf{x})], \quad (10)$$

where

$$\mathbf{s}(\mathbf{x}) = \frac{\mathbf{x} - \mathbf{r}}{\|\mathbf{x} - \mathbf{r}\|} \quad (11)$$

is the unit vector in the direction of \mathbf{x} from \mathbf{r} , and $\mathbf{e}_1(\mathbf{x}), \dots, \mathbf{e}_{d-1}(\mathbf{x})$ 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 $\lambda_s, \lambda_1, \dots, \lambda_{d-1}$, with

$$\lambda_s = 1 - \frac{1}{\Gamma(\mathbf{x})}, \quad (12)$$

$$\lambda_1, \dots, \lambda_{d-1} = 1 + \frac{1}{\Gamma(\mathbf{x})}. \quad (13)$$

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.

Huber et al. [7] proposed the following aggregation procedure. Let \mathbf{u}_i denote the individual modulations, with norms n_i . The final aggregate modulation \mathbf{u} is calculated as

$$\mathbf{u} = n_a \mathbf{u}_a, \quad (14)$$

where n_a and \mathbf{u}_a are the aggregate norm and direction.

The aggregate norm is computed as

$$n_a = \sum_{i=1}^N w_i n_i, \quad (15)$$

$$w_i = \frac{b_i}{\sum_{j=1}^N b_j}, \quad (16)$$

$$b_i = \prod_{1 \leq j \leq N, j \neq i} \Gamma_j(\mathbf{x}). \quad (17)$$

The above definition ensures that $\sum_{i=1}^N w_i = 1$, and $w_i \rightarrow 1$ 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 Huber et al. [7] calls “ κ -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 $R = [(\mathbf{x}^* - \mathbf{x})/\|\mathbf{x}^* - \mathbf{x}\|, \mathbf{e}_1, \dots, \mathbf{e}_{d-1}]$, where $\mathbf{e}_1, \dots, \mathbf{e}_{d-1}$ are again orthonormal 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 κ -space representation is

$$\boldsymbol{\kappa}_i = \frac{\arccos(\hat{\mathbf{u}}_i^{(1)})}{\sum_{m=2}^d \hat{\mathbf{u}}_i^{(m)}} \left[\hat{\mathbf{u}}_i^{(2)}, \dots, \hat{\mathbf{u}}_i^{(d)} \right]^T \in \mathbb{R}^{d-1}, \quad (18)$$

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 κ -space using the weights w_i calculated above

(19), transform it back to the R -frame (20), and finally transform it back to the original frame (21):

$$\boldsymbol{\kappa}_a = \sum_{i=1}^N w_i \boldsymbol{\kappa}_i \quad (19)$$

$$\hat{\mathbf{u}}_a = \left[\cos(\|\boldsymbol{\kappa}_a\|), \frac{\sin(\|\boldsymbol{\kappa}_a\|)}{\|\boldsymbol{\kappa}_a\|} \boldsymbol{\kappa}_a^T \right]^T \quad (20)$$

$$\mathbf{u}_a = R \hat{\mathbf{u}}_a. \quad (21)$$

As mentioned in Eq. 14, the final modulation is $\mathbf{u} = n_a \mathbf{u}_a$.

D.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 by Khansari-Zadeh and Billard [35] for a related but different type of modulation. Fig. 9, reproduced from the paper by Khansari-Zadeh and Billard [35, Fig. 7], shows the tail effect on the left and its removal on the right. This tail effect induces the placement of obstacles at the end of the ‘‘diagonal corridor’’ as seen in our straight-line deviation experiments (Fig. 5, left). If desired, the DS formulation can be modified to remove this effect.

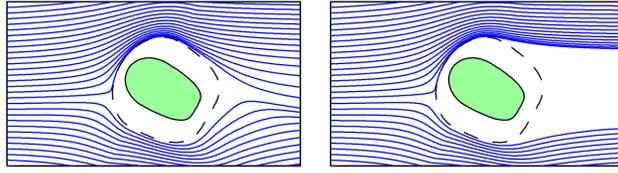


Figure 9: Tail effect (left) and its removal (right), reproduced from Fig. 7 by Khansari-Zadeh and Billard [35]. The target is on the far right side.

E RRT Algorithm Description and Sampling

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^* .

- 1 $\mathcal{T} \leftarrow \text{tree}(\text{root} = s_0)$;
- 2 $\text{success} \leftarrow \text{attempt-grow}(\mathcal{T}, \text{from} = s_0, \text{to} = s^*)$;
- 3 **while** *not* success **do**
- 4 $s \leftarrow \text{sample-configuration}()$;
- 5 $s_n \leftarrow \text{nearest-neighbor}(\mathcal{T}, s)$;
- 6 $\text{success} \leftarrow \text{attempt-grow}(\mathcal{T}, \text{from} = s_n, \text{to} = s)$;
- 7 **if** success **then**
- 8 $\text{success} \leftarrow \text{attempt-grow}(\mathcal{T}, \text{from} = s, \text{to} = s^*)$;
- 9 **return** $\text{path}(\mathcal{T}, \text{from} = s_0, \text{to} = s^*)$

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* $g = [s_1, s_2, s_3, \dots]$. 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 $\tau = \text{RRT}(s_0, s^*, g)$ 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 $\text{RRT}(s_0, s^*, g)$, 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. 7.

F MCMC Sampling Details

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 $\mathcal{N}(\mu = x, \sigma^2 = 0.1^2)$ 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. 5.1, 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 $[x_L, x_U]$, and $\sigma = 0.1(x_U - x_L)$ is used.

Each sampling run collected 10,000 samples, with the first 5,000 discarded as burn-in. On a consumer-grade computer with a single GeForce GTX 1080 GPU card (for neural network-based controllers), the sampling generally takes around 1 to 3 hours. The number of samples and burn-ins are selected fairly conservatively to ensure representativeness, as Fig. 10 plots the sampled behavior values in the chain for three analyses and confirms that these numbers are more than sufficient to ensure proper mixing. Note that ROCUS is designed to be an offline analysis tool as opposed to be used for real-time sample generation, and therefore several hours of runtime would be acceptable in most cases. Furthermore, MCMC sampling is embarrassingly parallel by simply using multiple chains concurrently, with the only overhead cost being the discarded burn-in samples.

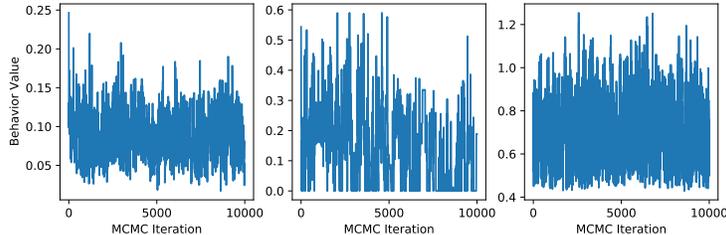


Figure 10: The sampled behavior values for three MCMC chains. From left to right, the three panels show DS min straight-line deviation on 2D navigation, RRT min straight-line deviation on 2D navigation and RL min end-effector movement on 7DoF arm reaching. The visualization confirms that 10,000 iterations with 5,000 burn-ins are more than sufficient to find representative samples.

G 2D Environment Details

In this domain, the environment is the area defined as $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. 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 $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{N_O} \in \mathbb{R}^2$, the environment is defined as

$$e(\mathbf{p}) = \sum_{i=1}^{N_O} \exp(-\gamma \|\mathbf{p} - \mathbf{p}_i\|_2^2), \quad (22)$$

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 $\eta = 0.9$. Fig. 11 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. Fig. 11 depicts a randomly selected assortment of 2D environments. These environments demonstrate the flexibility and diversity of the RBF environment definition.

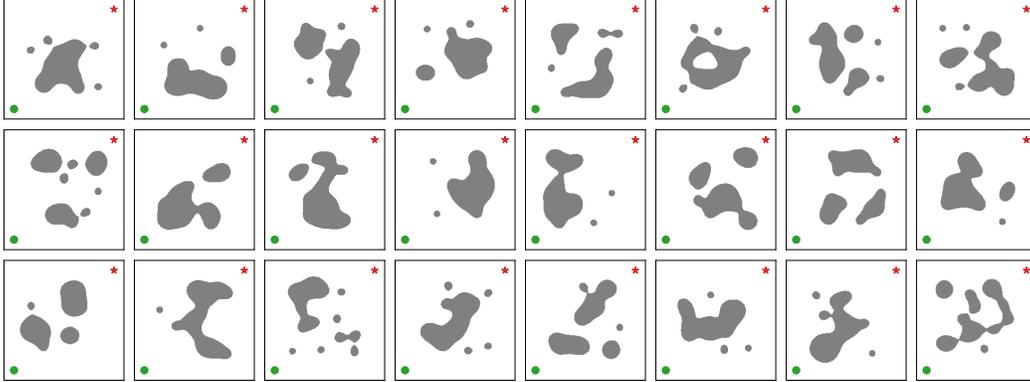


Figure 11: An assortment of randomly generated RBF 2D environments, providing a sense of the diversity generated with this formulation. The green dots are the environment starting points and the red stars are navigation targets. We show DS modulation for the first three environments in Fig. 13.

H Implementation Details of 2D Navigation Controllers

H.1 IL Controller

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 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 [36] and autonomous driving [37]. We follow the latter to treat this problem as classification with 100 bins in the $[0, 2\pi]$ range.

H.2 DS Controller

For the DS controller, there are two technical challenges in using the modulation [7] 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 in-

tersection 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. 12.

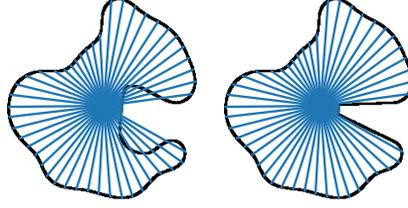


Figure 12: Left: an obstacle which is not star-shaped. Some radial lines extending from the obstacle’s reference point cross the boundary of the obstacle twice. Right: the same obstacle, modified to instead be star-shaped.

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

$$\Gamma(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{r}\|}{\|\mathbf{i} - \mathbf{r}\|}, \quad (23)$$

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 App. D.

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 Fig. 13.

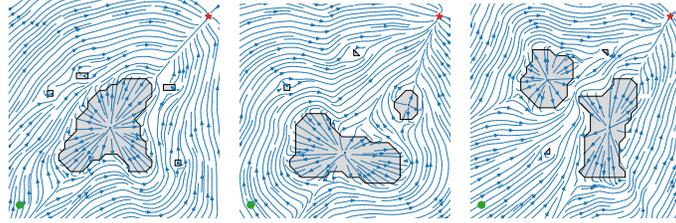


Figure 13: Streamlines showing the modulation effect of the dynamical system for three 2D navigation tasks. The environments correspond to the first three examples of Fig. 11. Green dots are starting positions and red stars are navigation targets.

I Additional Results for 2D Navigation

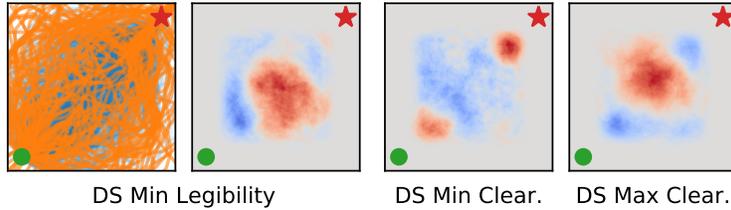


Figure 14: Left: trajectories and obstacle configurations from sampling minimal DS legibility. Right: obstacle configurations for minimizing and maximizing DS obstacle clearance. These examples show how obstacle positions affect the legibility and clearance behaviors.

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 often align to the target. 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. 14 present the samples. As expected, most trajectories take large detours due to the presence of obstacles in the center.

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. 14, 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 robot can avoid them by bearing hard left or right.

J Implementation Details of 7DoF Arm Reaching Controllers

J.1 RRT Controller

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 [38].

J.2 RL Controller

The RL controller implements the proximal policy gradient (PPO) algorithm [27]. 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.

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 learn the log variance for each of the 7 action dimensions. At test time, the policy deterministically outputs the mean action given a state.

J.3 DS Controller

For the DS controller in 7DoF arm reaching, we face the same challenges as in 2D navigation: defining an appropriate Γ -function for the obstacle configuration that holds the three properties introduced by Huber et al. [7] (listed in App. D). Additionally, the DS modulation technique does not consider the robot's morphology, end-effector shape, or workspace limits because it only modulates the state of a point-mass. Thus, we implement several adaptations. First, we modulate the 3D position of the tip of the end-effector. The desired velocity of the end-effector tip, given by the modulated linear controller, is then tracked by the 7DoF arm via the same position-level IK solver as the RRT controller.

Second, we used a support vector machine (SVM) to learn the obstacle boundary from a list of points in the obstacle and free spaces, an approach originally proposed by Mirrazavi Salehian et al. [31]. Then the decision function of the SVM is used as the Γ -function. As shown in Fig. 15, we discretize the 3D workspace of the robot and generate a dataset of points in the obstacle space as negative class and those in the free space as positive class.

Using the radial basis function (RBF) kernel $K(\mathbf{x}_1, \mathbf{x}_2) = e^{-\gamma\|\mathbf{x}_1 - \mathbf{x}_2\|^2}$, with kernel width γ , the SVM decision function $\Gamma(\mathbf{x})$ has the following form:

$$\Gamma(\mathbf{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b = \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma\|\mathbf{x} - \mathbf{x}_i\|^2} + b, \quad (24)$$

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

$$\nabla\Gamma(\mathbf{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i \frac{\partial K(\mathbf{x}, \mathbf{x}_i)}{\partial \mathbf{x}} = -\gamma \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma\|\mathbf{x} - \mathbf{x}_i\|^2} (\mathbf{x} - \mathbf{x}_i). \quad (25)$$

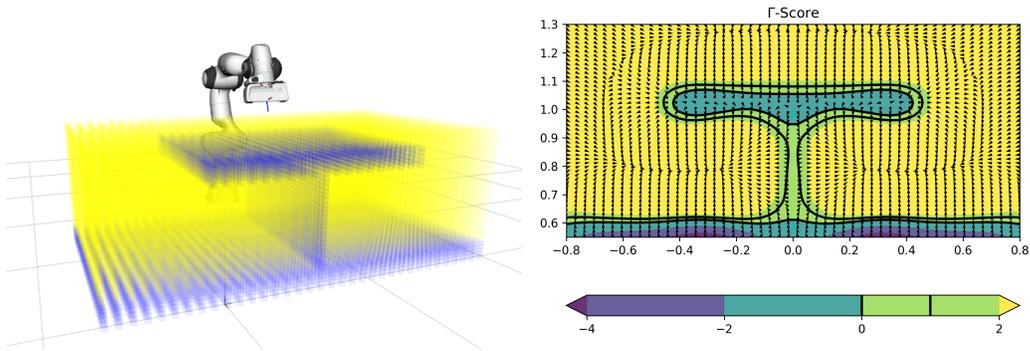


Figure 15: Left: the division of 3D space as either containing an obstacle or free space. This data is used to train an SVM, which acts as an interpolator. The classification scores of the SVM are used as the Γ function for this 3D reaching task. Right: a 2D slice showing the smoothed Γ scores.

In Eq. 24 and 25, \mathbf{x}_i ($i = 1, \dots, N_{sv}$) 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 `scikit-learn`. Using this learned Γ -function, Fig. 16 shows two examples of the modulated trajectory.

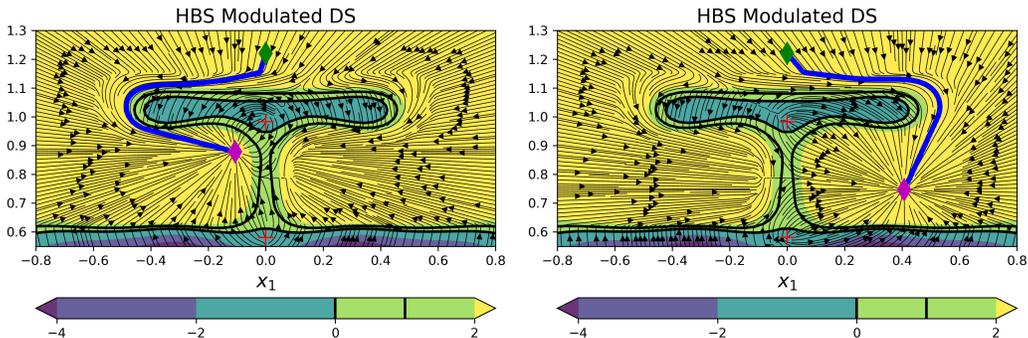


Figure 16: Cross-sections showing streamlines of the dynamical system modulation effect for two distinct targets in the 3D reaching task. Red crosses indicate reference points. Green diamond is the initial position of the end-effector for all experiments.

Finally, 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:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{u}_M(\mathbf{x}_t)\Delta t \quad (26)$$

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 [38].

K Additional Results for 7DoF Arm Reaching

Details on the DS Improvement The DS controller provides guarantees of convergence to a target in the space where modulation is applied (i.e. task-space in our experiments). To adopt this controller for obstacle avoidance with a robot manipulator, Huber et al. [7] simplifies the robot to a spherical shape with center at the end-effector of a 7DOF arm. This translates to considering the robot as a zero-mass point in 3D space but with the boundaries of the obstacles (described by Γ -functions) expanded by a margin with the size of the radius of the sphere.

Since the shape of the Franka robotic hand is rectangular ($6.3 \times 20.7 \times 14\text{cm}$) fitting a sphere with the radius of the longest axis will over-constrain the controller and drastically reduce the target regions inside the table dividers. We thus implemented the obstacle clearances by extruding the edges of the top table divider by half of the length of the robot’s end-effector (10cm) and the width of the divider by half of the height (7cm). Intuitively, this should be enough clearance to avoid the robot’s end-effector colliding with the table dividers. However, when coupling the DS controller with the IK solver to control the 7DoF arm, we noticed that the success rate was below 15%, whereas the success rate is 100% when controlling the end-effector only. We then sampled, via ROCUS, the target locations for the minimal final end-effector distance to target and noticed that all of the successful runs were located on the left-side of the partition (Fig. 6 center right).

Since the DS controller approach does not consider collision avoidance in joint-space, in a constrained environment, the robot’s forearm or elbow might get stuck on the edges of the table divider—even though the end-effector is avoiding collision. Due to the asymmetric kinematic structure of the robot arm, it is more prone to these situations on the right side of the table divider. Such an insight is not easy to discover as one must understand how the robot will behave in joint space based on its kinematic structure and the low-level controller used (position-based IK). We thus extended the edge extrusions to 20cm. This change improved the controller success rate and behavior drastically as shown in (Fig. 6 rightmost).

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, $V(\mathbf{x}) = -\max(\tilde{x}_1, 0)$, where $\tilde{x}_1 = x_1$ if target is on the left, or $\tilde{x}_1 = -x_1$ otherwise, and 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.

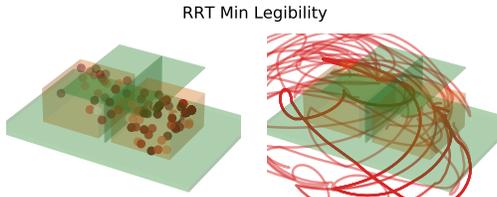


Figure 17: Posterior samples showing minimal legibility behavior for RRT.

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 do not use an optimal formulation [e.g. 39, 40] or perform post-hoc smoothing, the controller is expected to frequently exhibit low legibility. Fig. 17 plots the posterior target locations and trajectories. The target locations leading to illegible motions are spread out mostly uniformly on the right, but concentrated in far-back area on the left, consistent with our findings on the asymmetry of configuration space. The trajectory plot confirms the illegibility.

L Future Work

There are multiple directions to extend and complement ROCUS for better usability and more comprehensive functionality. First, 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* [33], 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, which sensor input (e.g. lidar or camera) is most important to the current action (e.g. swerving)? 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 [41, 42], but various issues with existing approaches have been raised [43, 44] and future research is 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.