How to Learn and Generalize From Three Minutes of Data: Physics-Constrained and Uncertainty-Aware Neural Stochastic Differential Equations

Franck Djeumou* **Cyrus Neary*** **Ufuk Topcu**
Center of Autonomy, University of Texas at Austin
{ fdjeumou, cneary, utopcu}@utexas.edu

Figure 1: With only three minutes of manually collected data, the proposed algorithms for training neural stochastic differential equations yield accurate models of hexacopter dynamics. When used for model-based control, they result in policies that track aggressive trajectories that push the hexacopter's velocity and Euler angles to nearly double their maximum values in the training dataset. Videos of experiments are available at https://tinyurl.com/29xr5vya.

Abstract: We present a framework and algorithms to learn controlled dynamics models using neural stochastic differential equations (SDEs)—SDEs whose drift and diffusion terms are both parametrized by neural networks. We construct the drift term to leverage a priori physics knowledge as inductive bias, and we design the diffusion term to represent a *distance-aware* estimate of the uncertainty in the learned model's predictions—it matches the system's underlying stochasticity when evaluated on states *near* those from the training dataset, and it predicts highly stochastic dynamics when evaluated on states *beyond* the training regime. The proposed neural SDEs can be evaluated quickly enough for use in model predictive control algorithms, or they can be used as simulators for model-based reinforcement learning. Furthermore, they make accurate predictions over long time horizons, even when trained on small datasets that cover limited regions of the state space. We demonstrate these capabilities through experiments on simulated robotic systems, as well as by using them to model and control a hexacopter's flight dynamics: A neural SDE trained using only three minutes of manually collected flight data results in a model-based control policy that accurately tracks aggressive trajectories that push the hexacopter's velocity and Euler angles to nearly double the maximum values observed in the training dataset.

Keywords: Neural SDE, Physics-Informed Learning, Data-Driven Modeling, Dynamical Systems, Control, Model-Based Reinforcement Learning

1 Introduction

Leveraging physics-based knowledge in learning algorithms for dynamical systems can greatly improve the data efficiency and generalization capabilities of the resulting models, even if the dynamics are largely unknown [1]. However, before such physics-constrained learning algorithms may be used

*These authors contributed equally.

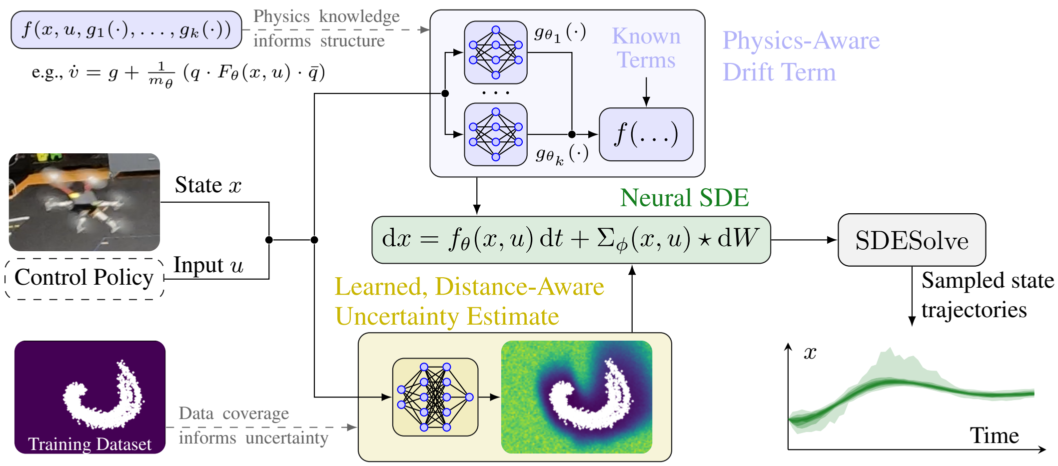

Figure 2: Overview. We propose neural stochastic differential equations (SDEs) to predict system dynamics. The SDE's drift and diffusion terms are both parametrized by neural networks. The drift term leverages a priori knowledge into the structure of the corresponding neural network. The diffusion term is trained to approximately capture the uncertainty in the model's predictions.

for model-based reinforcement learning (RL) or control, methods to estimate the uncertainty in the model's predictions are required [2, 3, 4, 5].

To build uncertainty-aware, data-driven models that also leverage a priori physics knowledge, we present a framework and algorithms to learn controlled dynamics models using *neural stochastic differential equations (SDEs)*—SDEs whose drift and diffusion terms are both parametrized by neural networks. Given an initial state and a control signal as inputs, the model outputs sampled trajectories of future states by numerically solving the SDE. Figure 2 illustrates an overview of the proposed approach, which comprises the following two major components.

Physics-constrained drift term. We represent the drift term as a differentiable composition of known terms that capture a priori system knowledge (e.g., the rigid body dynamics for the hexacopter from Figure 1), and unknown terms that we parametrize using neural networks (e.g., the nonlinear motor-to-thrust relationship and unmodeled aerodynamic effects). Even if the system's dynamics are largely unknown, this approach imposes useful inductive biases on the model: Unknown terms are parametrized separately and their interactions are defined explicitly.

Uncertainty-aware diffusion term. Meanwhile, we construct the diffusion term to provide a *distance-aware* estimate of the uncertainty in the learned model's predictions—the neural SDE's predictions should match the system's underlying stochasticity when it is evaluated on states similar to those from the training dataset, and they should be highly stochastic when it is evaluated on states beyond the dataset. In this work, we use Euclidean distance to evaluate this notion of similarity. To build such a diffusion term, we propose a novel training objective that encourages the entries in the diffusion matrix to have local minima at each of the training datapoints and to be strongly convex in a small surrounding neighborhood, while simultaneously ensuring that the resulting neural SDE's predictions fit the training data.

By leveraging the physics-based knowledge encoded in the drift term's structure, the proposed models are capable of making accurate predictions of the dynamics, even when they are trained on limited amounts of data and evaluated on inputs well beyond the regime observed during data collection. Furthermore, the proposed diffusion term not only allows the models to represent stochastic dynamics but also acts as a conservative estimate of the uncertainty in the model's predictions. Finally, the proposed neural SDEs can be evaluated quickly enough for use in online model-predictive control algorithms, or they can be used as offline simulators for model-based reinforcement learning.

We demonstrate these capabilities through experiments on simulated robotic systems, as well as by using them to model and control a hexacopter's flight dynamics (illustrated in Figure 1). The prediction accuracy of the proposed neural SDEs is an order of magnitude higher than that of a baseline model, and the neural SDEs also more reliably estimate the uncertainty in their own predictions. When applied to an offline model-based RL task, the control policies that result from our neural SDEs achieve similar performance to model-free RL policies, while requiring $30\times$ less data.

2 Related Work

Neural ordinary differential equations (ODEs) [6], which parametrize the right-hand side of a differential equation using a neural network, are a class of models that provide a natural mechanism for incorporating existing physics and engineering knowledge into neural networks [1, 7]. For example, many works use the Lagrangian, Hamiltonian, or Port-Hamiltonian formulation of dynamics to inform the structure of a neural ODE [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. Other works use similar physics-inspired network architectures to learn models that are useful for control [21, 22, 23, 24, 25, 26, 27, 28, 29].

However, these models are typically deterministic and cannot estimate their own uncertainty. By contrast, designing uncertainty-aware models is a primary focus of this work. On the other hand, neural SDEs have also been recently studied in the context of generative modeling, learning stochastic dynamics, and estimating the uncertainty in neural network parameters [30, 31, 32, 33, 34, 35, 36]. However, to the best of our knowledge, we are the first to propose the use of neural SDEs to capture model uncertainty while also leveraging a priori physics knowledge, and to apply them for model-based RL and control of dynamical systems.

Estimates of model uncertainty are crucial to preventing *model exploitation* in model-based reinforcement learning, particularly in the offline setting [2, 3, 4, 5]. Indeed, many model-based RL algorithms explicitly rely on measures of model uncertainty, or alternatively on some measure of the *distance* to the previously observed training data, during policy synthesis [37, 38, 3, 39, 4, 40, 41, 42, 43, 44]. One popular method to estimate the uncertainty of learned dynamics models is through *probabilistic ensembles* [3], which use ensembles of Gaussian distributions (whose mean and variance are parametrized by neural networks) to represent aleatoric and epistemic uncertainty. We similarly propose a method to learn uncertainty-aware models of dynamical systems that are useful for downstream control algorithms. However, in contrast with existing ensemble-based methods, the proposed neural SDEs are capable of leveraging a priori physics knowledge, they only require the training and memory costs associated with a single model, and they provide uncertainty estimates that are explicitly related to the distance of the evaluation point to the training dataset.

3 Our Approach: Physics-Constrained, Uncertainty-Aware Neural SDEs

Throughout the paper, we assume that a dataset $\mathcal{D} = \{\tau_1, \dots, \tau_{|\mathcal{D}|}\}$ of system trajectories is available in lieu of the system's model, where $\tau = \{(x_0, u_0), \dots, (x_{|\tau|}, u_{|\tau|})\}$, $x_i \in \mathcal{X}$ is the state of the system at a time $t_i(\tau)$, $u_i \in \mathcal{U}$ is the state-dependent control input applied at time $t_i(\tau)$, and $\mathcal{X}, \mathcal{U}$ are the state and input spaces, respectively. Given any current state value x_k at time t_k, and any control signal u or sequence of control inputs, we seek to learn a model to predict trajectories $x_{k+1}, \dots, x_{k+n_\mathrm{r}}$ of length n_r. Specifically, our objective is to train neural network models of dynamical systems that account for stochastic dynamics and measurement noise, that explicitly model prediction uncertainties, and that allow for the incorporation of a priori physics-based knowledge.

3.1 Using Neural SDEs to Model Systems with Unknown Dynamics

We propose to use neural SDEs to learn models of unknown dynamics, as opposed to learning ensembles of probabilistic models [45, 3] that directly map states and control inputs to distributions over future states. A neural SDE is an SDE whose drift and diffusion terms are parametrized by neural networks, and its state is a stochastic process that evolves according to

$$\mathrm{d}x = f_\theta(x, u)\,\mathrm{d}t + \Sigma_\phi(x, u) \star \mathrm{d}W, \tag{1}$$

where $f_\theta : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^{n_\mathrm{x}}$ and $\Sigma_\phi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{w}}$ are the drift and diffusion terms, respectively, and W is the n_w-dimensional Wiener process. The notation $\star$ indicates that the SDE is either in the Itô [46] or Stratonovich [47] form. The reader unfamiliar with these forms should feel free to ignore the distinction [48, 49, 30], which becomes an arbitrary modeling choice when f_θ, Σ_ϕ are learned. Under mild Lipschitz assumptions [50, 51] on f_θ, Σ_ϕ, we can efficiently sample a distribution of predicted trajectories via numerical integration of the neural SDE [52, 53]. Furthermore, we can incorporate physics knowledge into such models, discussed further in Section 3.3.

End-to-end SDE training. Given $\mathcal{D}$, we train the unknown functions f_θ and Σ_ϕ in an end-to-end fashion. For fixed parameters θ and ϕ, we integrate the neural SDE (1) to obtain a set of system trajectories over the time horizon $[t_i(\tau), t_{i+n_\mathrm{r}}(\tau)]$. We then compute the log-likelihood of the true trajectories in the dataset under the predicted trajectories, and use this to define the loss

$$\mathcal{J}_\mathrm{data} = \sum_{(x_i, u_i, \ldots, x_{i+n_\mathrm{r}}) \in \mathcal{D}} \frac{1}{n_\mathrm{p}} \sum_{j=i+1}^{n_\mathrm{r}} \sum_{p=1}^{n_\mathrm{p}} \left\| x_j^p - x_j \right\|^2, \tag{2}$$

$$\{x_{i+1}^p, \ldots, x_{i+n_\mathrm{r}}^p\}_{p=1}^{n_\mathrm{p}} = \mathrm{SDESolve}(x_i, u_i, \ldots, u_{i+n_\mathrm{r}-1}; f_\theta, \Sigma_\phi),$$

where $(x_i, u_i, \ldots, x_{i+n_\mathrm{r}}) \in \mathcal{D}$ denotes any sequence of transitions of length n_r from the dataset, the predicted trajectory $x_{i+1}^p, \ldots, x_{i+n_\mathrm{r}}^p$ is the p-th path sampled by any *differentiable* numerical integrator SDESolve, and n_p is the total number of sampled paths. We note that the expression in (2) corresponds to the log-likehood of the true state x_j with respect to a fixed-variance Gaussian distribution with mean given by the predicted state x_j^p. This formulation has been a common choice the literature [33, 30]. However, we note that other probability densities are also admissible and we pick the above distribution for simplicity in exposition. We then optimize the parameters θ and ϕ by minimizing the loss $\mathcal{J}_\mathrm{data}$ in (2) using either automatic differentiation through SDESolve or via the adjoint sensitivity method [33] for SDEs.

3.2 Using Neural SDEs to Represent Distance-Aware Estimates of Uncertainty

We emphasize that direct optimization of the model as presented in Section 3.1 would result in the diffusion approaching zero, if the drift term alone could be used to fit the training data well [31]. Instead, we propose to design, constrain, and train the diffusion term to represent modeling uncertainties. In particular, we propose an approach to learn a *distance-aware* diffusion term Σ_ϕ^dad that captures the true stochasticity of the dynamics when it is evaluated near points in the training dataset (in terms of Euclidean distance), while outputting highly stochastic predictions for points that are far from the dataset. A feature of our approach is that Σ_ϕ^dad is a neural network trained by locally sampling points near the training dataset, and yet its predictions respect useful global properties. Specifically, we translate this notion of distance-awareness into several mathematical properties that Σ_ϕ^dad must satisfy and propose a suitable loss function for its training. For notational simplicity, we use $[\mathrm{xu}]_i$ to denote the vector that results from concatenating x_i and u_i.

Bounded diffusion far from training dataset. On evaluation points sufficiently far from the training dataset, the entries in the diffusion term should saturate at maximum values. We accordingly propose to write $\Sigma_\phi^\mathrm{dad}([\mathrm{xu}]) \coloneqq \Sigma^\mathrm{max}([\mathrm{xu}]) \odot h_\phi(\eta_\psi([\mathrm{xu}]))$, where $\eta_\psi : \mathcal{X} \times \mathcal{U} \to [0, 1]$ is a parametrized scalar-valued function that encodes the notion of distance awareness, $h_\phi : [0, 1] \to [0, 1]^{n_\mathrm{x} \times n_\mathrm{w}}$ is a parameterized element-wise monotonic function, $\Sigma^\mathrm{max} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{w}}$ is a matrix specifying element-wise maximum values, and $\odot$ denotes the element-wise product.

Learning the system stochasticity near training dataset. On points included in the training dataset, Σ_ϕ^dad should not introduce any additional stochasticity related to model uncertainty. It should only capture the stochasticity of the dynamics that can be observed through the training data alone. Mathematically, we require that each entry of the diffusion term has a local minima at each pair (x_i, u_i) in $\mathcal{D}$. We impose this property on Σ_ϕ^dad by training η_ψ to have zero-gradient values at every point in the training dataset. This yields the loss function $\mathcal{J}_\mathrm{grad} = \sum_{[\mathrm{xu}]_i \in \mathcal{D}} \left\| \nabla_{[\mathrm{xu}]} \eta_\psi([\mathrm{xu}]_i) \right\|$.

Increasing diffusion values along paths that move away from the training dataset. As the query point moves away from the points in the training dataset, the magnitude of the diffusion term should monotonically increase. Let Γ be any path along which the distance from the current point to the nearest training datapoint always increases. Then, along Γ, the entries of Σ_ϕ^{dad} should monotonically increase. We enforce this property via local strong convexity constraints near the training dataset. Specifically, for every datapoint $(x_i, u_i) \in \mathcal{D}$ and a fixed radius $r > 0$, we design η_ψ to be strongly convex within a ball $\mathcal{B}_r(x_i, u_i) \coloneqq \{(x, u) \mid \|[\text{xu}] - [\text{xu}]_i\| \leq r\}$ with a strong convexity constant $\mu_i > 0$. That is, for any $(x, u), (x', u') \in \mathcal{B}_r(x_i, u_i)$, we desire that the constraint $\text{SC}_\psi(x, u, x', u') \geq 0$ holds, where SC_ψ is defined as:

$$\text{SC}_\psi(x, u, x', u') \coloneqq \eta_\psi([\text{xu}]') - \eta_\psi([\text{xu}]) - \nabla_{[\text{xu}]}\eta_\psi([\text{xu}])^\top([\text{xu}]' - [\text{xu}]) - \mu_i\|[\text{xu}]' - [\text{xu}]\|^2.$$

Now, to eliminate the strong convexity constants μ_i as tuneable hyperparameters, we parametrize a function $\mu_\zeta : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ using a neural network. Note that while μ_ζ is a continuous function over $\mathcal{X} \times \mathcal{U}$, we only ever evaluate it at points in the training dataset. This is effectively equivalent to learning separate values of μ_i for every $(x_i, u_i) \in \mathcal{D}$. We then define the loss functions

$$\mathcal{J}_{\text{sc}} = \sum_{(x_i,u_i)\in\mathcal{D}} \sum_{\substack{(x,u),(x',u') \\ \sim\mathcal{N}((x_i,u_i),r)}} \begin{cases} 0, \text{ if } \text{SC}_\psi(x, u, x', u') \geq 0 \\ (\text{SC}_\psi(x, u, x', u'))^2, \text{ otherwise} \end{cases} \quad \text{and } \mathcal{J}_\mu = \sum_{(x_i,u_i)\in\mathcal{D}} \frac{1}{\mu_\zeta(x_i, u_i)},$$

where $\mathcal{N}((x_i, u_i), r)$ is a Gaussian distribution with mean (x_i, u_i) and standard deviation r. Here, $\mathcal{J}_\mu$ is a regularization loss term that encourages high values of μ_i. This ensures that η_ψ reaches its maximum value of 1 close to the boundaries of the training dataset.

Modeling details. In practice, η_ψ has a sigmoid at its output, $\eta_\psi(x, u) \coloneqq \text{sigmoid}(\text{NN}_\psi(x, u))$. Furthermore, we define $h_\phi(z) \coloneqq \text{sigmoid}(W \text{sigmoid}^{-1}(z) + b)$, where $W, b \in \mathbb{R}^{n_{\text{x}}, n_{\text{w}}}$, and we restrict each entry of W to be greater than 1. Thus, $h_\phi(\eta_\psi(x, u)) = \text{sigmoid}(\text{NN}_\psi(x, u)W + b)$, which allows it to learn heterogeneous distributions of noise from the output of η_ψ.

Empirical illustration of η_ψ. To demonstrate the effectiveness of the designed constraints and loss functions, we train η_ψ to optimize $\lambda_{\text{grad}}\mathcal{J}_{\text{grad}} + \lambda_{\text{sc}}\mathcal{J}_{\text{sc}} + \lambda_\mu\mathcal{J}_\mu$ on two illustrative datasets of 2D trajectories. We parametrize η_ψ and μ_ζ as neural networks with swish activation functions, 2 hidden layers of size 32 each for η_ψ and size 8 each for μ_ζ. We pick the values $\lambda_{\text{grad}} = \lambda_{\text{sc}} = \lambda_\mu = 1$, and the ball radius $r = 0.05$. Figure 3 illustrates the learned η_ψ evaluated on the grid $[-0.2, 0.2] \times [-0.2, 0.2]$. We observe that η_ψ outputs low values near the training dataset, that these values increase as we move away from the dataset, and that the values are close to 1 when evaluated far from the dataset.

Figure 3: Output values of η_ψ after training on illustrative sample datasets in a 2D domain.

3.3 Leveraging A Priori Physics Knowledge in Neural SDEs

We represent the drift term f_θ in (1) as the composition of a known function – derived from a priori knowledge – and a collection of unknown functions that must be learned from data. That is, we write $f_\theta(x, u) \coloneqq F(x, u, g_{\theta_1}(\cdot), \ldots, g_{\theta_d}(\cdot))$, where F is a known differentiable function and $g_{\theta_1}(\cdot), \ldots, g_{\theta_d}(\cdot)$ are unknown terms within the underlying model. The inputs to these functions could themselves be arbitrary functions of the states and control inputs, or the outputs of the other unknown terms. Furthermore, by designing the element-wise maximum values $\Sigma^{\max}(x, u)$ for the diffusion term, we can encode prior notions of uncertainty into the neural SDE by specifying the maximum amount of stochasticity that the model will predict when evaluated on any given region of the state space. We provide several examples of such physics-based knowledge in Section 4. The complete neural SDE learning problem can now be formulated as

$$\underset{\theta_1,\ldots,\theta_d,\psi,\phi,\zeta}{\text{minimize}} \quad \lambda_{\text{data}}\mathcal{J}_{\text{data}} + \lambda_{\text{grad}}\mathcal{J}_{\text{grad}} + \lambda_{\text{sc}}\mathcal{J}_{\text{sc}} + \lambda_\mu\mathcal{J}_\mu. \tag{3}$$

4 Experimental Results

We evaluate our approach by comparing its performance for modeling and model-based control tasks against state-of-the-art techniques such as probabilistic ensembles [45, 3, 54], system identification-based algorithms [55, 56, 57], and neural ODEs [6, 1].

In all experiments, we use JAX [58] to implement and train the neural SDEs. We also use the same batch size and learning rate scheduler for the optimizer, the same neural network architecture for η_ψ, h_ϕ, and μ_ζ, and the same parameters $n_{\rm p}, \lambda_{\rm data}, \lambda_{\rm grad}, \lambda_{\rm sc}$. Only r, λ_μ, and $\Sigma^{\max}$ vary across experiments.

4.1 Spring-Mass-Damper: Generalization and Uncertainty Estimation with Limited Data

We start with the problem of modeling the dynamics of an uncontrolled spring-mass-damper system given 5 *noisy trajectories* of 5 seconds each. The equations of motion are given by the state $x = [q, \dot{q}]$ where q is the position of the mass and $\dot{q}$ is its velocity.

SDE model. We train the physics-informed model $\mathrm{d}x = [\dot{q}, g_{\theta_1}(x)]\mathrm{d}t + \sigma^{\max} \odot h_\phi(\eta_\psi(x)) \star \mathrm{d}W$, where the vector $\sigma^{\max}$ specifies the maximum level of stochasticity outside of the training dataset. We note that while this SDE model assumes $\ddot{q}$ is unknown, it takes advantage of the knowledge that the rate of change of q is $\dot{q}$.

Neural SDE improves uncertainty estimates and accuracy over probabilistic ensembles. As baselines for comparison, we train a neural ODE (which has the same architecture as the SDE model but excludes the diffusion term), and an ensemble of 5 probabilistic (Gaussian) models [54]. Figure 4 shows the prediction accuracy and uncertainty estimates of the learned models on a grid discretization of the state space. We use each point of the grid as an initial state for the models to predict 100 trajectories over a time horizon of 0.2 seconds. The first row shows that neural SDE and ODE models have high prediction accuracy. The neural SDE's accuracy decreases only slightly outside of the training dataset. On the other hand, for this low data regime, the probabilistic ensemble model is at least one order of magnitude less accurate than the neural SDE at points near the training dataset. The second row illustrates the variance of the model predictions, which we interpret as a measure of uncertainty. While the neural ODE has no notion of prediction uncertainty, the proposed neural SDE yields uncertainty estimates that agree with the availability of the training data. On the other hand, the probabilistic ensemble seems to provide high uncertainty estimates outside of the training dataset. However, near the dataset, there is no clear pattern as to when it will produce stochastic versus deterministic predictions.

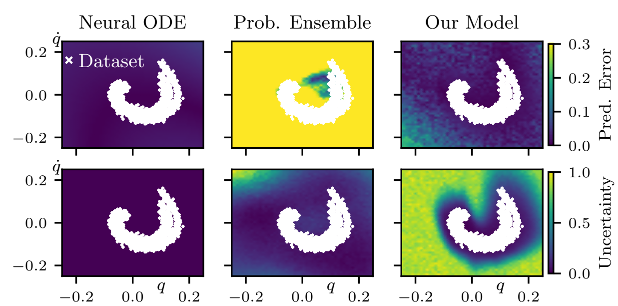

Figure 4: Model prediction errors and uncertainty estimates, when trained on the illustrated dataset.

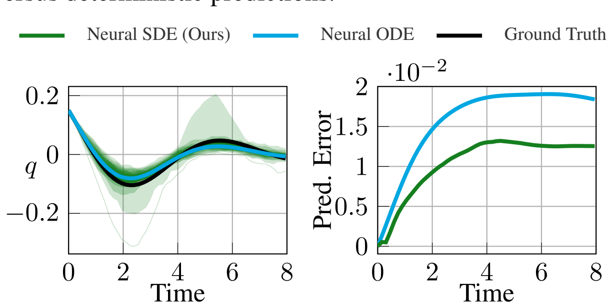

Figure 5: Neural SDE improves accuracy over neural ODE.

Neural SDE generalizes beyond the training dataset. Figure 5 shows the state trajectory predicted by the neural SDE and the neural ODE, from a given initial state. We observe that both models generalize well beyond the training dataset and are suitable for long-term prediction. However, the neural SDE's predictions are slightly more accurate than those of the neural ODE.

4.2 Cartpole Swingup: Offline Model-Based Reinforcement Learning with Neural SDEs

We now consider the cartpole swingup problem. The system state is defined by $x := [p_x, \dot{p}_x, \theta, \dot{\theta}]$ and the scalar control input u. Here, p_x is the position of the cart, θ is the angle of the pendulum, and u is the force applied to the cart.

Data collection. In contrast to the Mass-Spring-Damper experiments, we now utilize a moderate amount of data to ensure that the probabilistic ensemble methods are able to learn reasonably accurate models of the dynamics. Specifically, we build two training datasets, each of which consists of 100 trajectories of 200 system interactions. The first dataset, referred to as *random dataset*, is collected by applying random control inputs. The second dataset, referred to as *on-policy dataset*, is collected by applying inputs generated from a policy trained through Proximal Policy Optimization (PPO) [59] and 600000 interactions with the ground truth dynamics model.

Experimental setup. We consider two physics-constrained neural SDE models with different prior knowledge. The first model, simply referred to as neural SDE, is given by $\mathrm{d}x = [\dot{p}_x, f_{\theta_1}(x,u), \dot{\theta}, f_{\theta_2}(x,u)]\mathrm{d}t + \sigma^{\max} \odot h_\phi(\eta_\psi(x)) \star \mathrm{d}W$ while the second model, referred to as neural SDE with side info, exploits the known control-affine structure of the dynamics and is given by: $\mathrm{d}x = [\dot{p}_x, f_{\theta_1}(x) + g_{\theta_1}(\theta,\dot{\theta})u, \dot{\theta}, f_{\theta_2}(x) + g_{\theta_2}(\theta,\dot{\theta})u]\mathrm{d}t + \sigma^{\max} \odot h_\phi(\eta_\psi(x)) \star \mathrm{d}W$.

We train both neural SDE models, as well as an ensemble of probabilistic (Gaussian) models [54], using both the random and the on-policy datasets. For each learned model, we train a control policy via the PPO algorithm while using the model as an environment simulator. We compare the performance of the learned policies with the baseline policy used to generate the on-policy dataset. We note that the learned policies are trained on a dataset that is $30\times$ smaller than that required to train this baseline model-free RL policy. Figure 6 illustrates the mean episodic reward values achieved when evaluating the learned policies using the true dynamics (hatch pattern) vs using the learned dynamics models (grid pattern).

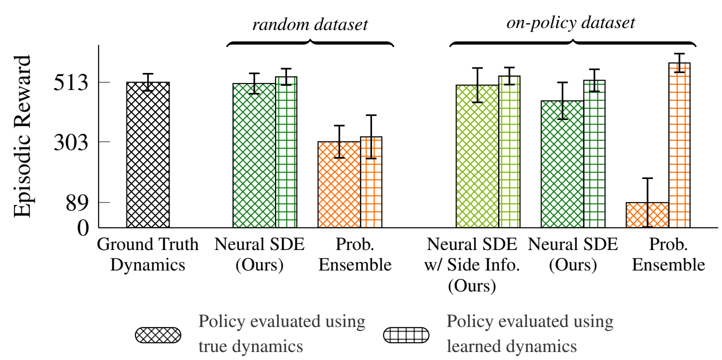

Figure 6: Reward of RL policies trained using learned dynamics.

Offline model-based RL using neural SDEs is as performant as model-free RL, while requiring $30\times$ fewer interactions with the environment. For the *random dataset*, the model-based policy trained using the neural SDE achieves comparable performance to the model-free baseline policy. By contrast, the policy trained using the probabilistic ensemble is significantly lower. For the *on-policy datset* (which is more limited in its coverage of the state-action space), the neural SDE results in a model-based policy that is less performant than the baseline policy. However, this gap is closed when considering the neural SDE that incorporates more physics knowledge (control-affine dynamics).

Neural SDEs are robust against model exploitation. For the *on-policy dataset*, the probabilistic ensemble results in a model-based policy that performs much worse when operating in the true environment than it does when using the learned model as a simulator. However, the neural SDE appears to be much more robust to this symptom of model exploitation. We note that many model-based RL algorithms are designed explicitly to prevent the policy optimization procedure from exploiting model inaccuracies. We did not employ such algorithms in this experiment but instead used an implementation of the PPO algorithm with default parameters [60].

4.3 Hexacopter System: Real-Time, Learning-Based Control with 3 Minutes of Data

We now demonstrate that by using only basic knowledge of rigid-body dynamics and 3 minutes of flight data, our approach bridges the sim-to-real gap and enables real-time control of a hexacopter.

Experimental setup. Our hexacopter uses a *CubePilot Orange* flight controller running PX4 [61], and a *Beelink MINIS 12* as the main computational unit. The CubePilot fuses measurements from the onboard IMU and motion capture system to estimate the state $x := [p_W, v_W, q_{WB}, \omega_B]$ at a frequency of 100 Hz. Here p_W and v_W are the position and velocity in the world frame, q_{WB} is the quaternion representing the body orientation, and ω_B is the angular rate in the body frame. The Beelink unit compiles our SDE models, solves a model predictive control (MPC) problem at each new state estimate, and sends back the resulting motor and desired angular rate commands.

Data collection: 3 minutes worth of data. We collect 3 system trajectories by manually flying the hexacopter via a radio-based remote controller. We store the estimated states x and input motor commands $u \in [0, 1]$[6] at a frequency of 100 Hz. We obtain a total of 203 seconds worth of flight data. An analysis shows that 95% of the data corresponds to the hexacopter operating below the speed of $1.71 \mathrm{m/s}$, absolute roll of $18°$, and absolute pitch of $13°$ while the maximum absolute speed, roll, and pitch attained are respectively 2.7 m/s, $23°$, and $23°$.

SDE model. The physics-constrained model leverages the structure of 6-dof rigid body dynamics while parametrizing the aerodynamics forces and moments, the motor command to thrust function, and (geometric) parameters of the system such as the mass and the inertia matrix:

$$\mathrm{d}\begin{bmatrix} v_W \\ \omega_B \end{bmatrix} = \begin{bmatrix} \frac{1}{m_\theta}\left(q_{WB}\left(T_\theta(u) + f_\theta^{\mathrm{res}}(x^{\mathrm{feat}})\right)\bar{q}_{WB}\right) + g_W \\ J_\theta^{-1}\left(M_\theta(u) + M_\theta^{\mathrm{res}}(x^{\mathrm{feat}})\right) - \omega_B \times J_\theta \omega_B \end{bmatrix} \mathrm{d}t + \sigma^{\max} \odot h_\phi(\eta_\psi(x^{\mathrm{feat}})) \star \mathrm{d}W,$$

where $x^{\mathrm{feat}} = [v_W, \omega_B]$, $\times$ denotes the cross product, $\bar{q}_{WB}$ is the conjugate of q_{WB}, the vector $\sigma^{\max}$ is the maximum diffusion calibrated by overestimating sensor noise, the variables m_θ and $J_\theta = \mathrm{diag}(J_\theta^{\mathrm{x}}, J_\theta^{\mathrm{y}}, J_\theta^{\mathrm{z}})$ represent the system mass and inertia matrix, the neural network functions f_θ^{res} and M_θ^{res} represent the residual forces and moments due to unmodelled and higher order aerodynamic effects, the parametrized functions T_θ and M_θ estimate the map between motor commands, thrusts, and moment values. g_W is the gravity vector. Besides, our model assumes the elementary knowledge that $\mathrm{d}p_W = v_W \mathrm{d}t$ and $\mathrm{d}q_{WB} = 0.5 q_{WB}[0, \omega_B]\mathrm{d}t$, which completes the full SDE model.

Neural SDE improves prediction accuracy over system identification while also providing uncertainty estimates. As a baseline model, we remove the diffusion term in our SDE model and use a nonlinear system identification (SysID) [57, 56] algorithm to learn the parametrized terms m_θ, J_θ, f_θ^{res}, M_θ^{res}, T_θ and M_θ. The nonlinear unknown functions are parametrized using a polynomial basis. Figure 7 shows the model predictions along a trajectory that was not included in the training dataset. We split the trajectory into segments of 1 second and perform open-loop prediction of the state evolution given the segment's initial state and the motor commands. The proposed neural SDE demonstrates improved accuracy and stability in comparison with the SysID baseline.

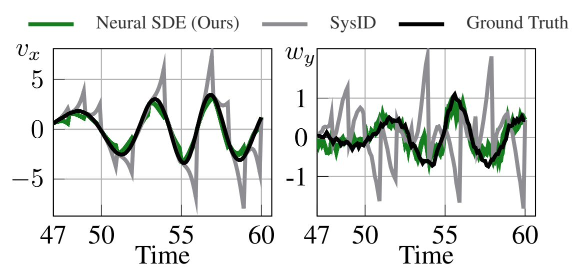

Figure 7: Our neural SDE achieves accurate and stable long-horizon predictions.

Neural SDE results in highly performant MPC even when operating beyond the training regime. Figure 8 shows the result of using the learned neural SDE in a MPC algorithm to track a lemniscate trajectory. The trajectory is obtained through generic minimum snap trajectory generation [62] without any knowledge of the hexacopter dynamics. Figure 8 demonstrates the ability of the neural SDE to generalize far beyond the limited training dataset, and to achieve high performance in terms of tracking accuracy when used for MPC. For example, for the first 10 seconds, the hexacopter reaches speed up to 1.7 m/s, yet the tracking error remains only 6 cm.

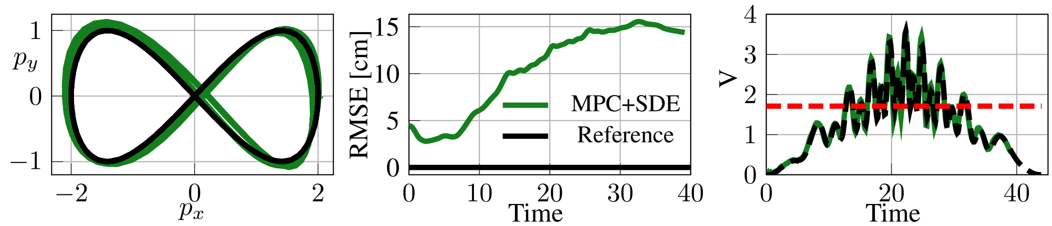

Figure 8: Tracking performance on the lemniscate trajectory. We obtain a root mean squared error (RMSE) of 15 cm despite operating at speeds up to 3.6 m/s and roll, pitch angles up to $32°$, nearly twice as high as the maximum values from the training dataset (shown by the dashed red line).

5 Discussion and Limitations

We present a framework and algorithms for learning uncertainty-aware models of controlled dynamical systems, that leverage a priori physics knowledge. Through experiments on a hexacopter and on other simulated robotic systems, we demonstrate that the proposed approach yields data-efficient models that generalize beyond the training dataset, and that these learned models result in performant model-based reinforcement learning and model predictive control policies.

While our formulation is general enough to easily extend to partial state observations, our current experiments use noisy observations of the system state. Future work will extend the proposed approach to learn dynamics from more varied observations of the state, e.g., from image observations. Finally, we note that the primary focus of this work is on the development of novel data-driven dynamics models. We already demonstrate that these models are effective for model-based reinforcement learning, however, we leave the development of algorithms that explicitly use the proposed uncertainty estimates for policy optimization to future work.

References

[1] F. Djeumou, C. Neary, E. Goubault, S. Putot, and U. Topcu. Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, 2022.

[2] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[3] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[4] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

[5] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[7] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[8] S. A. Desai, M. Mattheakis, D. Sondak, P. Protopapas, and S. J. Roberts. Port-hamiltonian neural networks for learning explicit time-dependent dynamical systems. *Physical Review E*, 104(3):034312, 2021.

[9] S. Eidnes, A. J. Stasik, C. Sterud, E. Bøhn, and S. Riemer-Sørensen. Pseudo-hamiltonian neural networks with state-dependent external forces. *Physica D: Nonlinear Phenomena*, 446: 133673, 2023.

[10] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020.

[11] A. Zhu, P. Jin, and Y. Tang. Deep hamiltonian networks based on symplectic integrators. *arXiv preprint arXiv:2004.13830*, 2020.

[12] C. Allen-Blanchette, S. Veer, A. Majumdar, and N. E. Leonard. Lagnetvip: A lagrangian neural network for video prediction. *arXiv preprint arXiv:2010.12932*, 2020.

[13] Y. D. Zhong, B. Dey, and A. Chakraborty. Extending lagrangian and hamiltonian neural networks with differentiable contact models. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., 2021.

[14] Y. D. Zhong, B. Dey, and A. Chakraborty. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Proceedings of the 3rd Conference on Learning for Dynamics and Control*, volume 144. PMLR, 2021.

[15] Y. D. Zhong, B. Dey, and A. Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[16] M. A. Roehrl, T. A. Runkler, V. Brandtstetter, M. Tokic, and S. Obermayer. Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics. *IFAC-PapersOnLine*, 53(2):9195–9200, 2020.

[17] C. Neary and U. Topcu. Compositional learning of dynamical system models using port-hamiltonian neural networks. In *5th Annual Learning for Dynamics & Control Conference*, 2023.

[18] M. Finzi, K. A. Wang, and A. G. Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[19] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[20] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

[21] X.-Y. Liu and J.-X. Wang. Physics-informed dyna-style model-based deep reinforcement learning for dynamic control. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2255).

[22] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural lander: Stable drone landing control using learned dynamics. In *International Conference on Robotics and Automation*, pages 9784–9790, 2019.

[23] S. S.-E. Plaza, R. Reyes-Báez, and B. Jayawardhana. Total energy shaping with neural interconnection and damping assignment - passivity based control. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168. PMLR, 2022.

[24] L. Furieri, C. L. Galimberti, M. Zakwan, and G. Ferrari-Trecate. Distributed neural network control with dependability guarantees: a compositional port-hamiltonian approach. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168. PMLR, 2022.

[25] J. Wong, V. Makoviychuk, A. Anandkumar, and Y. Zhu. Oscar: Data-driven operational space control for adaptive and robust robot manipulation. In *2022 International Conference on Robotics and Automation*, 2022.

[26] K. Menda, J. K. Gupta, Z. Manchester, and M. J. Kochenderfer. Structured mechanical models for efficient reinforcement learning. In *Workshop on Structure and Priors in Reinforcement Learning, International Conference on Learning Representations*, 2019.

[27] J. K. Gupta, K. Menda, Z. Manchester, and M. Kochenderfer. Structured mechanical models for robot learning and control. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120. PMLR, 2020.

[28] T. Duong and N. Atanasov. Hamiltonian-based neural ode networks on the se (3) manifold for dynamics learning and control. In *Robotics: Science and Systems (RSS)*, 2021.

[29] M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2019.

[30] P. Kidger. *On neural differential equations*. PhD thesis, University of Oxford, 2021.

[31] W. Xu, R. T. Q. Chen, X. Li, and D. Duvenaud. Infinitely deep bayesian neural networks with stochastic differential equations. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151. PMLR, 2022.

[32] L. Kong, J. Sun, and C. Zhang. SDE-net: Equipping deep neural networks with uncertainty estimates. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119. PMLR, 2020.

[33] X. Li, T.-K. L. Wong, R. T. Chen, and D. K. Duvenaud. Scalable gradients and variational inference for stochastic differential equations. In *Symposium on Advances in Approximate Bayesian Inference*. PMLR, 2020.

[34] J. Jia and A. R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[35] L. Yang, T. Gao, Y. Lu, J. Duan, and T. Liu. Neural network stochastic differential equation models with applications to financial data forecasting. *Applied Mathematical Modelling*, 115: 279–299, 2023.

[36] P. Kidger, J. Foster, X. Li, and T. J. Lyons. Neural sdes as infinite-dimensional gans. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021.

[37] G. An, S. Moon, J.-H. Kim, and H. O. Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., 2021.

[38] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn. Combo: Conservative offline model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., 2021.

[39] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[40] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. In *International Conference on Learning Representations*, 2017.

[41] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[42] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[43] J. Jeong, X. Wang, M. Gimelfarb, H. Kim, B. abdulhai, and S. Sanner. Conservative bayesian model-based value expansion for offline policy optimization. In *The 11th International Conference on Learning Representations*, 2023.

[44] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

[45] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. volume 30. Curran Associates, Inc., 2017.

[46] K. Itô. *On stochastic differential equations*. Number 4. American Mathematical Soc., 1951.

[47] R. L. Stratonovich. A new representation for stochastic integrals and equations. *Siam Journal on Control*, 4:362–371, 1966.

[48] N. G. Van Kampen. Itô versus stratonovich. *Journal of Statistical Physics*, 24:175–187, 1981.

[49] S. Massaroli, M. Poli, S. Peluchetti, J. Park, A. Yamashita, and H. Asama. Learning stochastic optimal policies via gradient descent. *IEEE Control Systems Letters*, 6:1094–1099, 2021.

[50] B. Øksendal. *Stochastic differential equations*. Springer Berlin Heidelberg, 2003.

[51] H. Kunita. *Stochastic flows and stochastic differential equations*, volume 24. Cambridge university press, 1997.

[52] G. N. Milstein. *Numerical integration of stochastic differential equations*, volume 313. Springer Science & Business Media, 1994.

[53] P. E. Kloeden, E. Platen, and H. Schurz. *Numerical solution of SDE through computer experiments*. Springer Science & Business Media, 2002.

[54] L. Pineda, B. Amos, A. Zhang, N. O. Lambert, and R. Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *arXiv preprint arXiv:2104.10159*, 2021.

[55] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[56] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings. Mathematical, Physical, and Engineering Sciences*, 474, 2017.

[57] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton. Pysindy: a python package for the sparse identification of nonlinear dynamics from data. *arXiv preprint arXiv:2004.08424*, 2020.

[58] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

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

[60] A. Serrano-Muñoz, N. Arana-Arexolaleiba, D. Chrysostomou, and S. Bøgh. skrl: Modular and flexible library for reinforcement learning. *arXiv preprint arXiv:2202.03825*, 2022.

[61] L. Meier, D. Honegger, and M. Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.

[62] D. Mellinger and V. R. Kumar. Minimum snap trajectory generation and control for quadrotors. *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.