

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

## Supplementary Material

### A Implementation and Modeling Details

All numerical experiments were implemented using the python library JAX [58], in order to take advantage of its automatic differentiation and just-in-time compilation features. We use Python 3.8.5 for the experiments and train all our models on a laptop computer with an Intel i9-9900 3.1 GHz CPU with 32 GB of RAM and a GeForce RTX 2060, TU106. We provide the code for all experiments in the supplementary material with instruction on how to reproduce the results.

**Training optimizer hyperparameters.** We use the *Adam* optimizer [63] for all optimization problems. This includes when training the neural ODE model, the probabilistic ensemble model, and the system identification-based model. We use the default hyperparameters for the optimizer, except for the learning rate, which we linearly decay from 0.01 to 0.001 over the first 10000 gradient steps. We use early stopping criteria for all our experiments. We use a batch size of 512 for the neural ODE, SDE models, and the system identification-based model. Instead, we use a batch size of 32 for the ensembles of probabilistic models.

**Model design.** As specified in Section 4, across experiments, we use the same batch size and learning rate scheduler for the optimizer, the same neural network architecture for  $\eta_\psi$ ,  $h_\phi$ , and  $\mu_\zeta$ , and the same parameters  $n_p$ ,  $\lambda_{\text{data}}$ ,  $\lambda_{\text{grad}}$ ,  $\lambda_{\text{sc}}$ . Specifically, we parametrize  $\eta_\psi$  as a feedforward neural network with swish activation functions and 2 hidden layers of size 32 each while the parameters  $W, b$  of  $h_\phi$  are matrices of size corresponding to the dimension of  $\sigma^{\text{max}}$ . We parametrize  $\mu_\zeta$  as a feedforward neural network with tanh activation functions and 2 hidden layers of size 8 each. For the loss function penalty terms, we use  $\lambda_{\text{data}} = 1.0$  for the data loss,  $\lambda_{\text{grad}} = 0.01$  for the zero-gradient loss, and  $\lambda_{\text{sc}} = 0.01$  for the strong convexity loss. We use  $n_p = 1$  for training the neural SDE model. For the ensembles of probabilistic models, we always use 5 models in the ensemble, where each model is a Gaussian with mean and variance parametrized by feedforward neural networks with 2 hidden layers of size 64 each and silu activation functions.

## B Supplementary Mass-Spring-Damper Details and Results

The equations of motion are given by the state  $x := [x_1, x_2] = [q, \dot{q}]$  with  $\dot{x}_1 = x_2$  and  $m\dot{x}_2 = -bx_2 - kx_1$ , where  $q$  is the position of the mass,  $m$  is the mass,  $b$  is the damping coefficient, and  $k$  is the spring constant. We consider the case where  $m = 1$ ,  $b = 0.5$ , and  $k = 1$ ; all the quantities being in the international system of units.

**Data collection: Noisy and extremely scarce amount of data.** We collect two dataset  $\mathcal{D}_1$  and  $\mathcal{D}_2$  of 5 *trajectories* each. The trajectories are obtained from the known dynamics with the initial positions randomly sampled in the top right quadrant  $[0.1, 0.1] \times [0.05, 0.15]$  for  $\mathcal{D}_1$  and in the more broad region  $[-0.1, 0.1] \times [-0.1, 0.1]$  for  $\mathcal{D}_2$ . Each trajectory has a length of 5 seconds and are integrated through the Euler method with a discrete step size of 0.01 second. Besides, we add a zero-mean Gaussian noise with standard deviation of  $[0.005, 0.01]$  to each state measurement.

Specifically, the first dataset will be used to show that our neural SDE framework provides interpretable uncertainty estimate and better prediction accuracy than Gaussian ensemble in the extreme low and non-diverse data regime. The second dataset will be used to show that our neural SDE model improves prediction accuracy over neural ODE when the dataset is sufficiently diverse even in the low data regime.

**Benchmark models.** We assume that the dynamics of the mass-spring-system are unknown and we use the proposed neural SDE framework to train predictive models from the training dataset  $\mathcal{D}_1$  and  $\mathcal{D}_2$ . Our neural SDE model has the following structure:

$$dx = [x_2, f_\theta(x)]dt + \sigma^{\max} \odot h_\phi(\eta_\psi(x)) \star dW,$$

where  $f_\theta$  is a feedforward neural network with  $\tanh$  activation functions and 2 hidden layers of size 4 and 16, respectively. The vector  $\sigma^{\max} := [\sigma_1^{\max}, \sigma_2^{\max}] = [0.001, 0.02]$  provides the desired diffusion values outside of the training dataset.

We compare our neural SDE model with a learned neural ODE and an ensemble of probabilistic (Gaussian) models. The neural ODE model is trained with the same architecture as the neural SDE model without the diffusion term. For training the neural SDE and neural ODE models, we use Euler-Maruyama and Euler methods as the SDEsolve algorithms, respectively, with a step size of 0.01 second and a time horizon of 0.5 seconds. Further, we use  $\lambda_\mu = 1$  for encouraging large strong convexity coefficients and a vector of ball radius  $r = 0.05$  to locally enforce the strong convexity property.

**Neural SDE generalizes beyond the training dataset..** Figure 5 shows the state evolution of the neural SDE and neural ODE models trained on the more diverse dataset  $\mathcal{D}_2$  and evaluated for some initial condition outside the training dataset. We can observe that the SDE generalizes well beyond the training dataset and is suitable for long-term prediction. On the plot representing the evolution of  $\dot{q}$  as a function of time, we observe that the noise at the beginning is high as we start from a point outside of the training dataset. However, as we move closer to the training dataset with  $q$  and  $\dot{q}$  close to zero, the model becomes more confident and the noise decreases. We emphasize that we do not show the trained Gaussian ensemble model on this plot as it just diverge over 0.3 seconds of integration time.

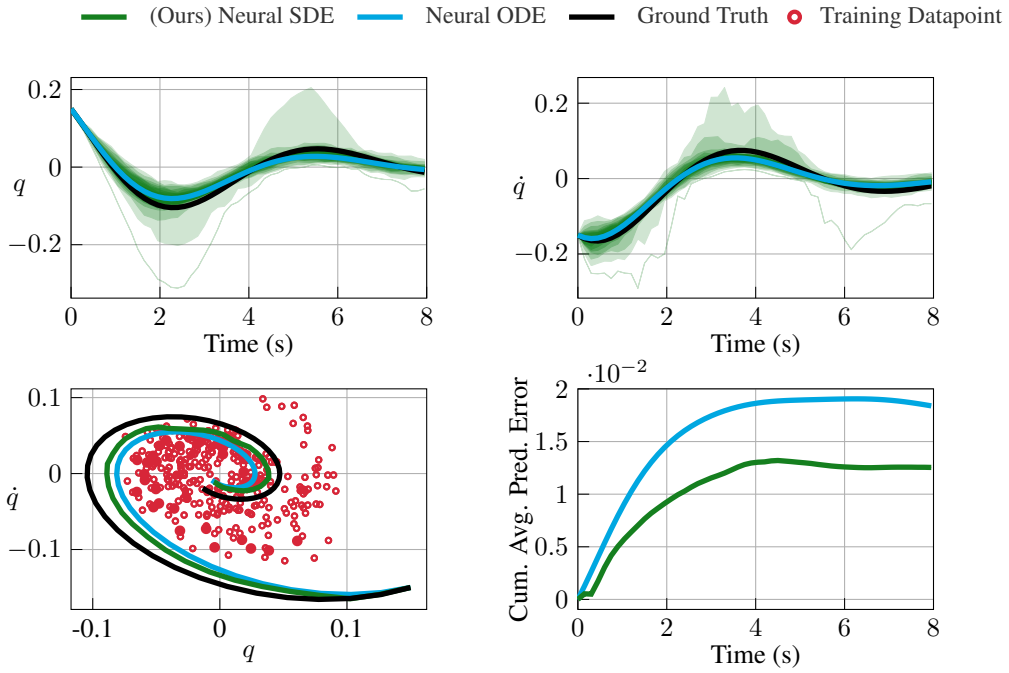


Figure 9: Prediction of the neural SDE and neural ODE models over a time horizon of 8 seconds for an initial condition  $x_{\text{init}} = [0.15, -0.15]$  outside the training dataset. The neural SDE generalizes well beyond the training dataset while providing accurate coverage of the groundtruth trajectory and improving accuracy over the neural ODE model.

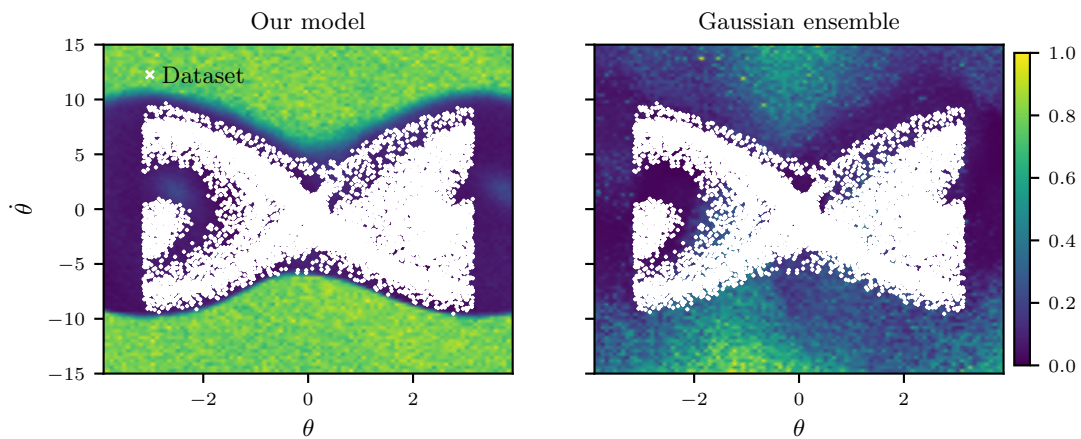


Figure 10: Model uncertainty estimates when trained on the *on-policy dataset*.

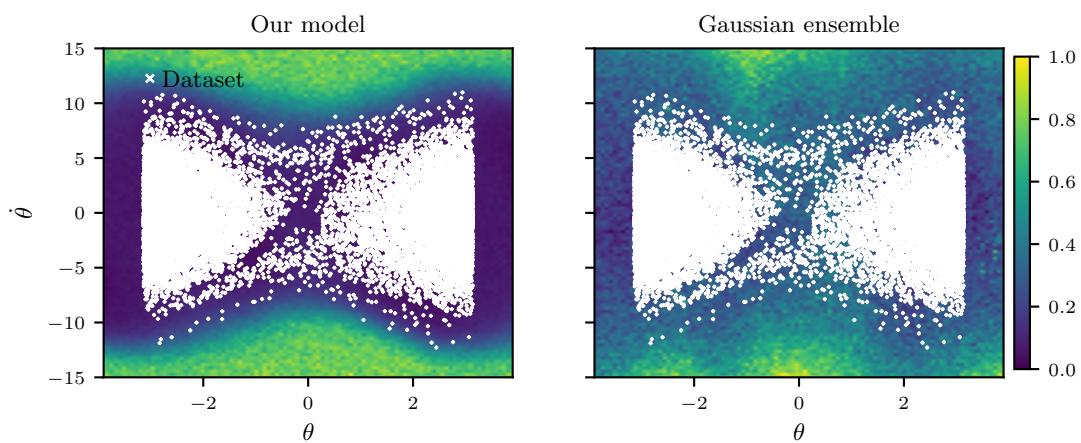


Figure 11: Model uncertainty estimates when trained on the *random dataset*.

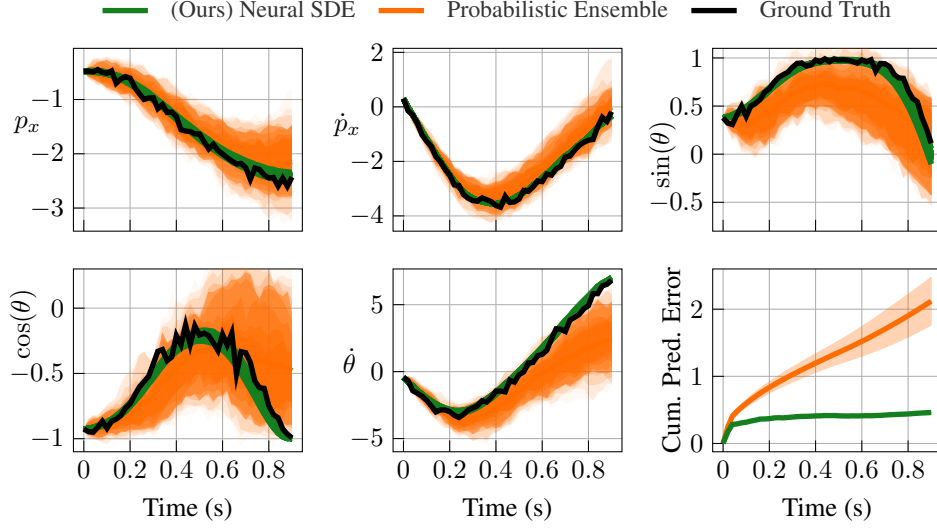


Figure 12: State evolution predicted by the neural SDE, in comparison with the predictions of the probabilistic ensemble.

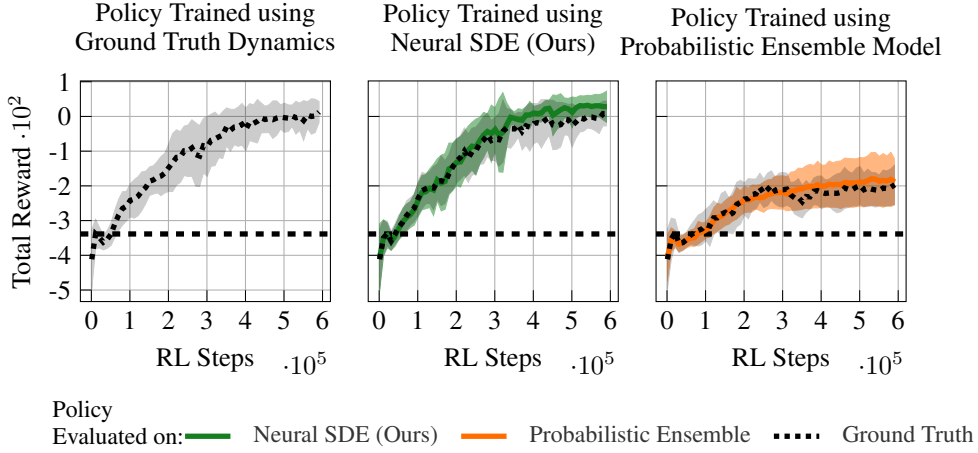


Figure 13: Mean episodic reward achieved by policies trained using PPO, while using the learned dynamics models as environment simulators. Training a policy using our proposed neural SDE model achieves identical reward to as when using the ground truth dynamics, but requires  $30\times$  fewer environment interactions. Left: Policy trained through interactions with the ground truth environment dynamics. Middle: Policy trained through interactions with the neural SDE (ours). Right: Policy trained through interactions with a probabilistic ensemble. The horizontal dotted line illustrates the reward achieved by a model-free approach that interacts directly with the ground truth dynamics, but whose number of environment interactions is restricted to that used to train the dynamics models.

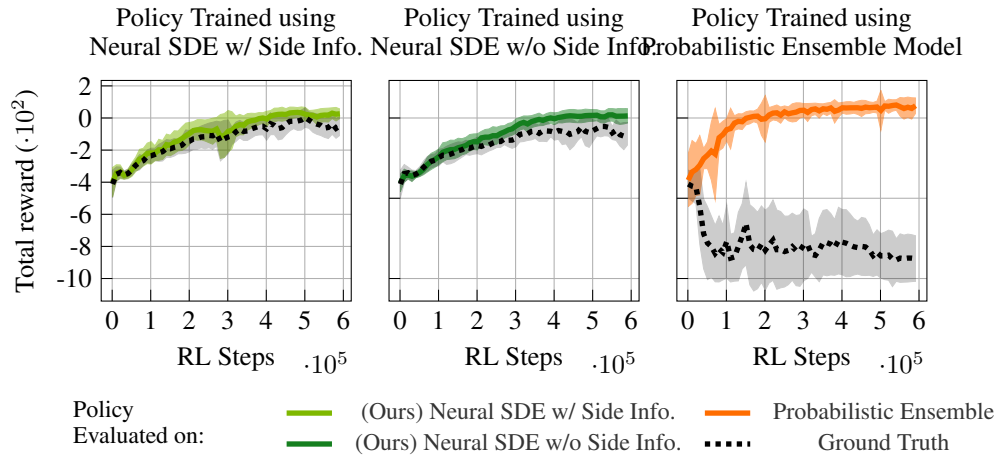


Figure 14: Mean episodic reward achieved by policies trained using PPO, while using learned dynamics models as environment simulators.

## D Supplementary Hexacopter Details and Results

This numerical experiment shows that with basic knowledge of rigid-body dynamics as prior physics knowledge for our neural SDE, we can learn accurate and uncertainty-aware predictive models for an hexacopter from only 3 minutes of manual flight. Then, using our learned model in a model predictive control (MPC) framework, we show incredible tracking performance on aggressive trajectories, despite how the reference trajectories push the hexacopter to operate far beyond what was seen during training.

The custom-built hexacopter has a *CubePilot Cube Orange* as flight controller running PX4 firmware [61]. The hexacopter is equipped with 920KV brushless motors, 10 inch and two-bladed propellers, and it features the *DJI F550* frame, which has a 550mm diagonal motor to motor distance.

The full state of the hexacopter is given by  $x = [p_x, p_y, p_z, v_x, v_y, v_z, q_w, q_x, q_y, q_z, \omega_x, \omega_y, \omega_z]$ , where  $p_W = [p_x, p_y, p_z]$  is the position in the world frame,  $v_W = [v_x, v_y, v_z]$  is the velocity in the world frame,  $q_{WB} = [q_w, q_x, q_y, q_z]$  is the unit quaternion representing the body orientation,  $\omega_B = [\omega_x, \omega_y, \omega_z]$  is the angular rate in the body frame, and the world and body frames follow respectively the traditional East-North-Up and Forward-Left-Up shown in Figure ?? . The state is estimated at a frequency of 100 Hz using the PX4 implementation of an Extended Kalman Filter that fuses the measurements from the onboard IMU and our Vicon motion capture system. Besides the CubePilot that handles the state estimation and motor control, the main computational unit is a *Beelink MINIS 12*. Its primary task is to compile our neural SDE models implemented in JAX, receive each new state estimate from the CubePilot, solve the stochastic NMPC, and send back the resulting motor commands and desired angular rate to the CubePilot for low-level motor control. In software in the loop simulation, we could track desired trajectories via sending motor commands directly output from our NMPC. However, due to the latency during hardware experiments, we instead send desired angular rates from the NMPC to the CubePilot, which then uses a PI controller to track the desired angular rates.

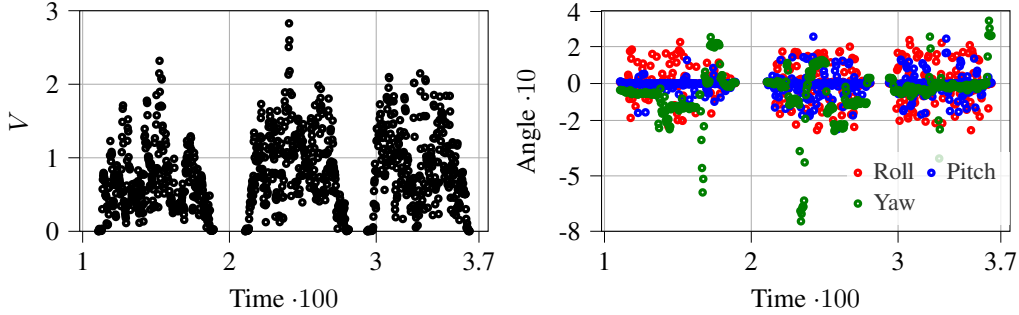


Figure 15: The velocity magnitude and Euler angles attained by the hexacopter during data collection. The hexacopter mostly operates in the low speed and low Euler angles regime.

**Data Collection: 3 Minutes worth of Data.** We are interested in learning a predictive model for the hexacopter’s dynamics that can be used to autonomously track aggressive trajectories. To this end, we collect 3 system trajectories by manually flying the hexacopter via a radio-based remote controller. During data collection, we store the estimated state  $x$  at a frequency of 100 Hz, as well as the desired input commands  $u = [u_1, u_2, u_3, u_4, u_5, u_6]$  sent to the motors. Figure 15 shows the velocity magnitude and euler angles from the collected dataset. We obtain a total of 203 seconds worth of flight data, with the 3 trajectories being respectively 73, 66, and 64 seconds long. Besides, Figure 15 shows that 95% of the collected data corresponds to the hexacopter operating below the speed of 1.71m/s, absolute roll of  $18^\circ$ , absolute pitch of  $13^\circ$ , and absolute yaw of  $24^\circ$ . Instead, the maximum absolute speed, roll, pitch, and yaw attained are respectively 2.7 m/s,  $23^\circ$ ,  $23^\circ$ , and  $80^\circ$ .

**Benchmark Models.** We use the proposed neural SDE framework to train a model of the hexacopter dynamics with the limited dataset described above. Our neural SDE model takes advantage of the general structure of 6-dof rigid body dynamics while having as unknown terms: 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. Specifically, our physics-informed neural SDE model is given by:

$$\mathrm{d} \begin{bmatrix} p_W \\ v_W \\ q_{WB} \\ \omega_B \end{bmatrix} = \begin{bmatrix} v_W \\ \frac{1}{m_\theta} (q_{WB} (T_\theta(u) + f_\theta^{\text{res}}(x^{\text{feat}})) \bar{q}_{WB}) + g_W \\ \frac{1}{2} q_{WB} \omega_B \\ J_\theta^{-1} (M_\theta(u) + M_\theta^{\text{res}}(x^{\text{feat}})) - \omega_B \times J_\theta \omega_B \end{bmatrix} \mathrm{d}t + \sigma^{\text{max}} \odot h_\phi(\eta_\psi(x^{\text{feat}})) \star \mathrm{d}W, \quad (4)$$

where  $x^{\text{feat}} = [v_W, \omega_B]$ ,  $\times$  denotes the cross product,  $\bar{q}_{WB}$  is the conjugate of  $q_{WB}$ , the product  $qv$  between a quaternion  $q$  and a vector  $v$  is define as the quaternion product between  $q$  and the 4-D vector  $[0; v]$ , the vector  $\sigma^{\text{max}} = [1, 1, 1, 10, 10, 10, 1, 1, 1, 1, 50, 50, 50] \cdot 10^{-3}$  is the maximum diffusion term, the variables  $m_\theta$  and  $J_\theta = \text{diag}(J_\theta^x, J_\theta^y, J_\theta^z)$  represent the system mass and inertia matrix, the neural network functions  $f_\theta^{\text{res}}$  and  $M_\theta^{\text{res}}$  represent the residual forces and moments due to unmodelled and higher order aerodynamic effects, the parametrized functions  $T_\theta$  and  $M_\theta$  provide estimate of the motor command to thrust and moment values, and  $g_W = [0, 0, -9.81]^\top$  is the gravity vector. Specifically, we parametrize  $f_\theta^{\text{res}}$ ,  $M_\theta^{\text{res}}$  as feedforward neural networks with tanh activation functions and 2 hidden layers of size 8 and 16, respectively. The motor thrust forces and moments are learned via  $[T_\theta^\top, M_\theta^\top]^\top = [0, 0, T_\theta^z, M_\theta^x, M_\theta^y, M_\theta^z]^\top = A_\theta^{\text{mix}} [T_\theta^{\text{mot}}(u_1), \dots, T_\theta^{\text{mot}}(u_6)]^\top$ , where  $A_\theta^{\text{mix}}$  is a  $6 \times 6$  matrix of learnable parameters constrained by the geometry of the hexacopter, and  $T_\theta^{\text{mot}}$  is a parametrized function that maps the motor commands to the thrust forces. We use polynomial functions for  $T_\theta^{\text{mot}}$  and empirically found that a degree of 1 as  $T_\theta^{\text{mot}}(z) = \alpha_\theta z + \beta_\theta$  works particularly well for control purpose compared to higher order polynomials. We emphasize the diffusion term on  $p_W$  and  $q_{WB}$  are low as their dynamics are known and the noise in the estimation will come from integrating the noisy velocity and angular rate components. For training the neural SDE model, we use the derivative-free Milstein method as the SDESolve algorithm with a step size of 0.05 second and a time horizon of 1 second. Further, we use  $\lambda_{\text{sc}} = 1$  for encouraging large strong convexity coefficients and a vector of ball radius  $r = 0.1$  to locally enforce the strong convexity property.

To illustrate the prediction accuracy of our model, we compare it with a system identification-based approach that uses the same formulation as our SDE model (4) but without the diffusion term and the residual neural network terms  $f_\theta^{\text{res}}$  and  $M_\theta^{\text{res}}$ . Precisely, with system identification, we seek to identify all the parameters  $m_\theta$ ,  $J_\theta$ ,  $A_\theta^{\text{mix}}$ ,  $\alpha_\theta$ , and  $\beta_\theta$  by estimating  $\dot{x}$  from the dataset using finite difference and then solving a least square problem to fit the system's differential equation to the data.

**Neural SDE improves prediction accuracy over system identification while also providing uncertainty estimates.**

**Nonlinear Model Predictive Control.** We seek to use our learned SDE for autonomous control of the hexacopter. To this end, we employ a receding horizon model predictive control approach. Such approach uses the learned SDE model to predict future system trajectories over a fixed time horizon, and then optimize the control inputs to minimize a cost function that penalizes the control effort and deviation from a reference trajectory. For numerical optimization, we discretize the state and control inputs into  $n_r = 20$  equal time intervals over the horizon  $H = 1$  second, yielding a constrained optimization problem of the following form at each state measurement  $x_t = x_{\text{init}}$ :

$$\underset{u_1, \dots, u_{n_r}}{\text{minimize}} \quad \mathbb{E} \left[ \sum_{k=1}^{n_r} (x_k - x_k^{\text{ref}}) Q (x_k - x_k^{\text{ref}}) + u_k R u_k \right] \quad (5)$$

$$\text{subject to} \quad \{x_1^p, \dots, x_{n_r}^p\}_{p=1}^{n_p} = \text{SDESolve}(x_0, u; (4)), \quad (6)$$

$$x_0 = x_{\text{init}}, \quad u_1, \dots, u_{n_r} \in [0, 1], \quad (7)$$

where  $x_k^{\text{ref}}$  is the reference state at time  $t_k = t + kH/n_r$ , we use  $q - q^{\text{ref}}$  to denote the term  $(q(q^{\text{ref}})^{-1})_{xyz}$ , the positive definite matrices  $Q$  and  $R$  penalize the deviation from the reference trajectory and the control effort, respectively, and  $n_p$  is the number of particles used for the SDE solver. For all our experiments, we used  $n_p = 1$ ,  $Q = \text{diag}(100, 100, 200, 5, 5, 10, 1, 1, 100, 1, 1, 1)$ , and  $R = \text{diag}(1, 1, 1, 1, 1, 1)$ . Besides, we solve the above optimization problem using an adaptive learning rate, Nesterov acceleration-based projected gradient descent; all implemented in JAX.



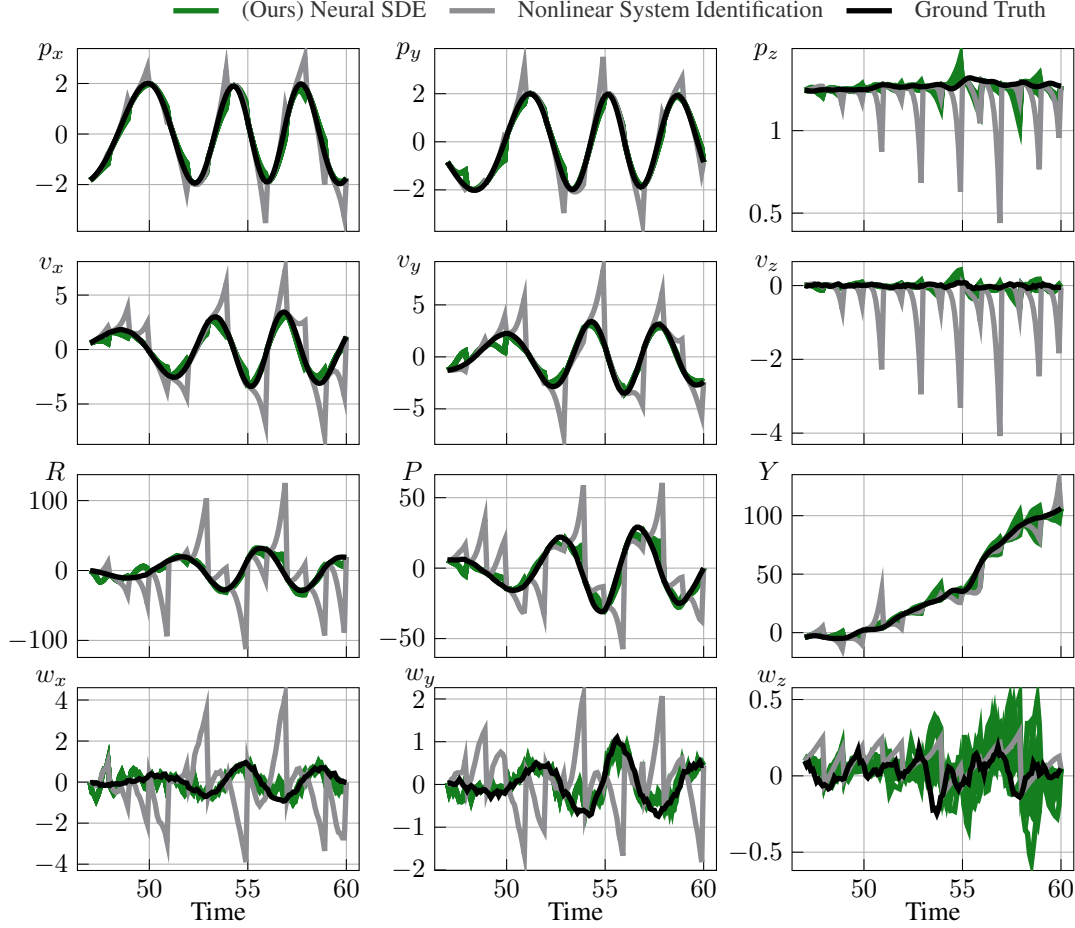


Figure 16: Time evolution of the hexacopter state predicted by the learned SDE model and the vanilla system identification-based model. The quantity  $R, P, Y$  denote the roll, pitch, and yaw angles, respectively, with unit in degrees.

Figure 17 and Figure 18 show the results of our experiments on a circle trajectory and lemniscate trajectory, respectively. These reference trajectories are obtained by minimum snap trajectory generation [14] without any prior knowledge of the hexacopter dynamics. For both trajectories, we show the time evolution of the velocity, roll, pitch, yaw, and the tracking accuracy during autonomous control. These plots demonstrate the ability of our learned SDE to generalize far beyond what it has been trained on. In fact, we can observe that the hexacopter must reach velocity up to 3.6 m/s, roll and pitch angles up to  $32^\circ$ , and yaw angle up to  $120^\circ$ , in order to track the reference trajectories. We emphasize how these values are outside of the training data regime as shown and detailed in the data collection section. Despite this, our learned SDE is able to generalize to these extreme conditions and achieves a high tracking accuracy of 20 cm and 15 cm for the circle and lemniscate trajectories, respectively. We also note that the tracking accuracy is not uniform across the trajectory, and it is lower when the hexacopter is moving faster. This is expected as the hexacopter is more difficult to control when it is moving faster. Besides, the performance of our control approach is further displayed on the plot showing the evolution of the altitude, where an altitude error of less than 5 cm is achieved.

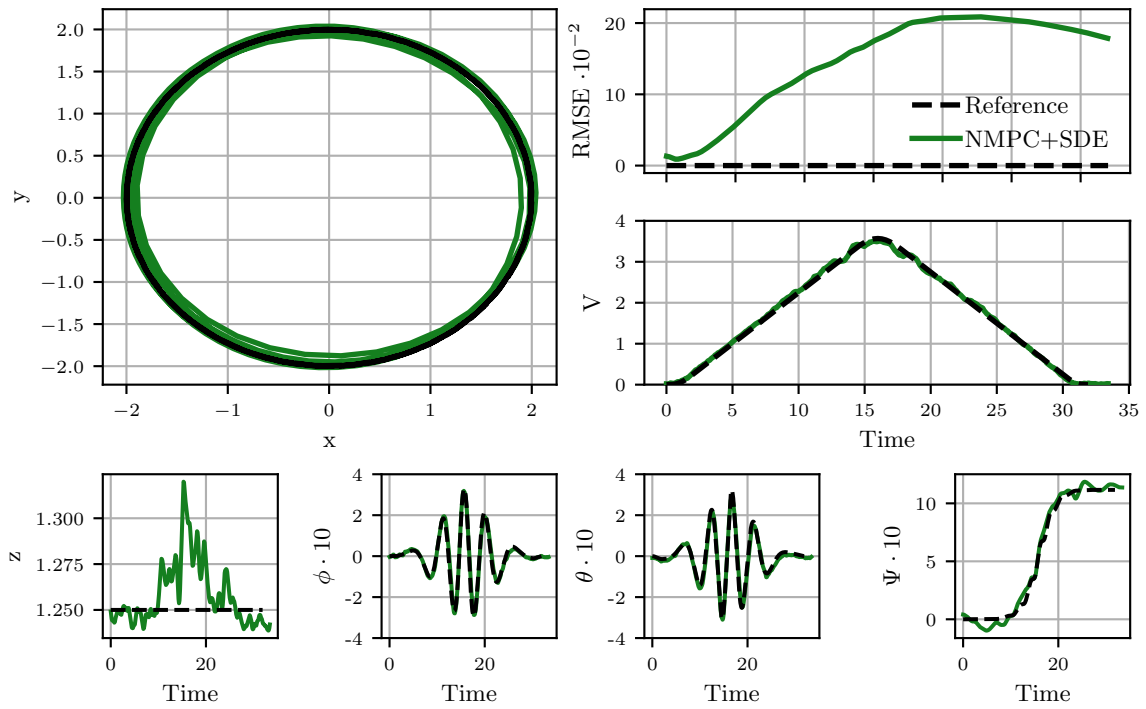


Figure 17: Tracking performance achieved on the circle trajectory. The NMPC based on our learned SDE achieves an RMSE of 20 cm for a 35 seconds trajectory while the hexacopter reaches speed up to 3.6 m/s, roll and pitch angles up to  $32^\circ$ , and yaw angle up to  $120^\circ$ .

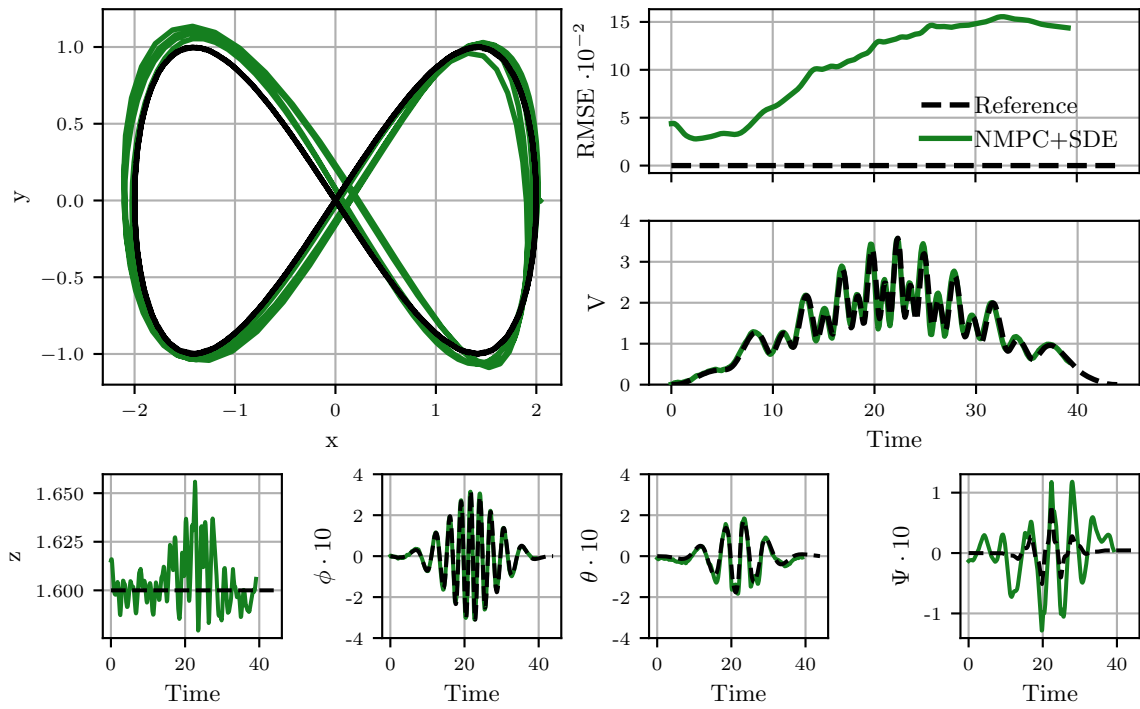


Figure 18: Tracking performance achieved on the lemniscate trajectory. The NMPC based on our learned SDE achieves an RMSE of 15 cm for a 40 seconds trajectory while the hexacopter reaches speed up to 3.4 m/s, roll angle up to  $32^\circ$ , pitch angle up to  $19^\circ$ , and yaw angle up to  $13^\circ$ .