Supplementary Material
No Need for Interactions: Robust Model-Based Imitation Learning using Neural ODE

A. Proof of Proposition 3.2

Eq.(16) from Proposition 3.1 can be rewritten by substituting Eq.(5) as follows:

$$\hat{\pi}_\phi(\boldsymbol{\nu}(t), \boldsymbol{x}(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} \tag{19}$$

Then, as defined in Proposition 3.2, replaces $\boldsymbol{x}(t)$ inside $\hat{\pi}_\phi$ and the term related to positive feedback gain $\boldsymbol{K_n}$ with $\boldsymbol{x}'(t) \sim \mathcal{N}(\boldsymbol{x}(t), \sigma_x)$. Such replacement is not applied on $\hat{G}_\theta^{-1}$ and $\hat{a}_\theta$ because the noise is injected outside of the system dynamics (θ). The resulting equation can be expressed as below:

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}'(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} \tag{20}$$

Since the noised state $\boldsymbol{x}'(t)$ is sampled from Gaussian distribution, we discuss all three possible cases as follows:
(The following operation for $\boldsymbol{x}(t)$ is element-wise, in order to simplify the derivation steps, we treat $\boldsymbol{x}(t)$ as a one dimension vector)

- $\boldsymbol{x}'(t) = \boldsymbol{x}(t)$: This implies $\boldsymbol{x}'(t)$ is state with no noise. Therefore, Eq.(20) is identical to Eq.(16) as below:

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))[\dot{\boldsymbol{x}}_r(t+\Delta t) - \hat{a}_\theta(\boldsymbol{x}(t))] \tag{21}$$

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \boldsymbol{u}_{ndi}(t) + \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{u}_{extra}(t) \tag{22}$$

  Where $\boldsymbol{u}_{extra}(t) \triangleq (\boldsymbol{K_n} \cdot 0)$ represents additional control effort and will be zero in this case.

- $\boldsymbol{x}'(t) > \boldsymbol{x}(t)$: This implies $\boldsymbol{x}(t)' = \boldsymbol{x}(t) + \delta\boldsymbol{x}(t)$, where $\delta\boldsymbol{x}(t) > 0$ represents a small disturbance. Then substituting this representation into Eq.(20) and rearranging the equation as follows,

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}(t) - \delta\boldsymbol{x}(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} \tag{23}$$

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} - \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t) \tag{24}$$

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \boldsymbol{u}_{ndi}(t) + \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{u}_{extra}(t) \tag{25}$$

  Where $\boldsymbol{u}_{extra}(t) = (-\boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t))$ and will be negative in this case.

- $\boldsymbol{x}'(t) < \boldsymbol{x}(t)$: This implies $\boldsymbol{x}'(t) = \boldsymbol{x}(t) - \delta\boldsymbol{x}(t)$. Then applying the same procedures as the case discussed above,

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}(t) + \delta\boldsymbol{x}(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} \tag{26}$$

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \hat{G}_\theta^{-1}(\boldsymbol{x}(t))\{\boldsymbol{K}_n[\boldsymbol{x}_r(t+\Delta t) - \boldsymbol{x}(t)] - \hat{a}_\theta(\boldsymbol{x}(t))\} + \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t) \tag{27}$$

$$\hat{\pi}_\phi(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \boldsymbol{u}_{ndi}(t) + \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{u}_{extra}(t) \tag{28}$$

  Where $\boldsymbol{u}_{extra}(t) = (\boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t))$ and will be positive in this case.

Based on the three cases discussed above, the additional control term $\boldsymbol{u}_{extra}(t)$ can be redefined as follows,

$$\boldsymbol{u}_{extra}(t) = \boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t) \cdot sgn(\boldsymbol{x}(t) - \boldsymbol{x}'(t)) \tag{29}$$

$$\approx \boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t) \cdot sgn(\boldsymbol{x}_r(t) - \boldsymbol{x}'(t)) \tag{30}$$

$$= \boldsymbol{K_s} \cdot sgn(\boldsymbol{\sigma}(\boldsymbol{x}'(t))) \tag{31}$$

$$\hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{u}_{extra}(t) = \hat{G}_\theta^{-1}(\boldsymbol{x}(t)) \cdot \boldsymbol{K_s} \cdot sgn(\boldsymbol{\sigma}(\boldsymbol{x}'(t))) = \boldsymbol{u}_{smc}(t) \tag{32}$$

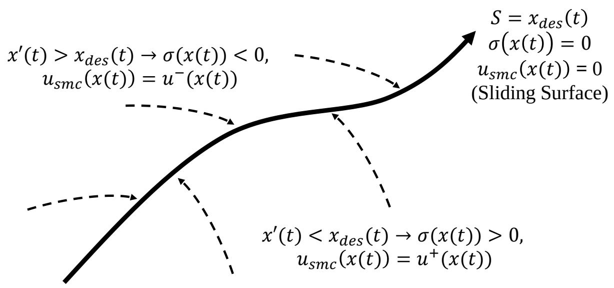
$S = x_{des}(t)$
$\sigma(x(t)) = 0$
$u_{smc}(x(t)) = 0$
(Sliding Surface)
$x'(t) > x_{des}(t) \rightarrow \sigma(x(t)) < 0,$
$u_{smc}(x(t)) = u^{-}(x(t))$
$x'(t) < x_{des}(t) \rightarrow \sigma(x(t)) > 0,$
$u_{smc}(x(t)) = u^{+}(x(t))$

Figure 4. Ancillary SMC Concept Illustration. The center solid curve represents the desired trajectory surface (S) where a closed loop system should slide along with. When system state is on the sliding surface, the SMC switching function $\sigma(\boldsymbol{x}(t))$ is zero, so the ancillary control effort $\boldsymbol{u}_{smc}$ becomes zero. In the case that the system state deviates from the surface S (that is on the dash curves) because of disturbances, the ancillary control effort $\boldsymbol{u}_{smc}$ will be decided based on the switching function $\sigma(\boldsymbol{x}(t))$ in order to drive the noised state $\boldsymbol{x}'(t)$ back to the desired sliding surface $\boldsymbol{S} = \boldsymbol{x}_{des}(t)$.

Where $\boldsymbol{K_s} = \boldsymbol{K_n} \cdot \delta\boldsymbol{x}(t) > 0$ and its value will be decided by $\hat{\pi}_{\phi}$ in order to drive the noised state $\boldsymbol{x}'(t)$ back to non-noised state $\boldsymbol{x}(t)$ as close as possible. The derivation from Eq.(29) to Eq.(30) is based on the assumption that the training loss ($\mathcal{L}_{\phi}(\tau)$) of the original learned NDI control policy $\hat{\pi}_{\theta}$ from Proposition 3.1 is almost zero. Under such a condition, $\hat{\pi}_{\theta}$ will always drives the system state $\boldsymbol{x}(t)$ to follow the desired state $\boldsymbol{x}_r(t+\Delta t)$ closely, which implies $\boldsymbol{x}(t) \approx \boldsymbol{x}_r(t)$ under ideal environment (without disturbances). The derivation of Eq.(32) relies on Eq.(8)

By substituting Eq.(32) back to Eq.(22), (25) and (28), the refined control policy $\hat{\pi}'_{\phi}$ can expressed as:

$$\hat{\pi}'_{\phi}(\boldsymbol{\nu}'(t), \boldsymbol{x}'(t)) \approx \boldsymbol{u}_{ndi}(t) + \boldsymbol{u}_{smc}(t) = \boldsymbol{u}_{rn}(t) \tag{33}$$

The approximation sign in Eq.(33) will be established if the training loss ($\mathcal{L}'_{\phi}$) of the refined policy $\hat{\pi}'_{\phi}$ approaches a very small number. Based on the above discussions, Proposition 3.2 is proven.

B. Derivation of Ancillary Sliding Mode Controller

If the environment is ideal (without disturbance), by applying Eq.(6), the NDI control law $\boldsymbol{u}_{ndi}$ will drive the system state $\boldsymbol{x}(t)$ to follow the desired trajectory surface $\mathcal{S} = \boldsymbol{x}_{des}(t)$ accurately. However, in case of dynamics mismatch or environmental noise, the controlled trajectories may deviate from the desired surface $\mathcal{S}$. Therefore, an ancillary controller would be helpful if it is able to bring the deviated state back to the desired surface based on the deviation metric $\sigma(\boldsymbol{x}(t)) = \boldsymbol{x}_{des}(t) - \boldsymbol{x}(t)$.

According to [27], the classic definition of SMC law is expressed as follows:

$$\boldsymbol{u}_{smc}(\boldsymbol{x}(t)) = \begin{cases} \boldsymbol{u}^{+}(\boldsymbol{x}(t)), & \text{if } \sigma(\boldsymbol{x}(t)) > 0 \\ \boldsymbol{u}^{-}(\boldsymbol{x}(t)), & \text{if } \sigma(\boldsymbol{x}(t)) < 0 \end{cases} \tag{34}$$

Where the choose of $\boldsymbol{u}^{+}(\boldsymbol{x}(t))$ should lead $\dot{\sigma}(\boldsymbol{x}(t)) < 0$, and $\boldsymbol{u}^{-}(\boldsymbol{x}(t))$ would lead $\dot{\sigma}(\boldsymbol{x}(t)) > 0$. In addition, Eq.(34) can be rewritten as a more compact form as: $\boldsymbol{u}_{smc}(\boldsymbol{x}(t)) = \boldsymbol{K_s} \cdot sgn(\sigma(\boldsymbol{x}(t))$, where $\boldsymbol{K_s} > 0$ and its value depends on $|\sigma(\boldsymbol{x}(t)|$. Fig.4 provides graphic illustration of the SMC concept as discussed above.

By applying Eq.(34) to the noised system, the derivative of the Lyapunov function $\dot{V}(\sigma(\boldsymbol{x}(t))) = \sigma(\boldsymbol{x}(t))^T \dot{\sigma}(\boldsymbol{x}(t))$ would be negative, which is the sufficient condition of global stability for a closed loop system [27].

C. Additional Information for the Experiments

This section provides additional information for the experiments described in the main essay. Sec.C.1 illustrates the training loss and imitation loss of the proposed RMBIL framework. Sec.C.2 describes the implementation details for the proposed framework and hyperparameters list used in the experiments. Sec.C.3 shows the screenshots for the uneven environments used for the robustness evaluation experiments.

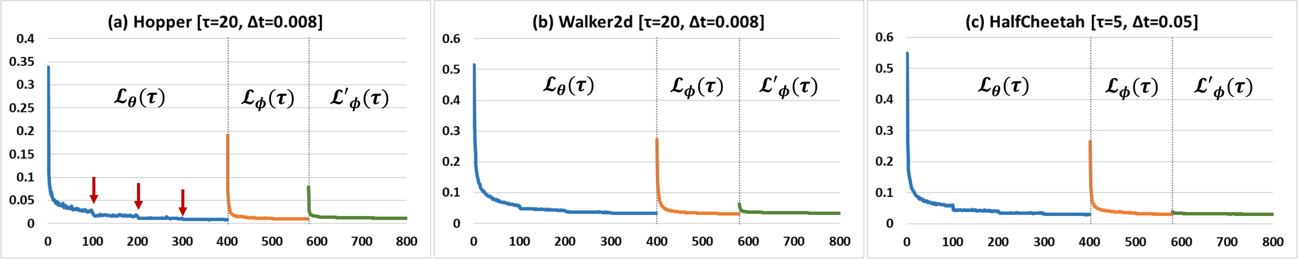

Figure 5. Training loss of Dynamics-Controller modules (Neural ODE) versus training epochs. The x-axis is the number of training epochs. The y-axis is the training loss with respect to the specified prediction horizon τ. Blue curves (first) are the training loss for dynamics model $\hat{f}_\theta$. Orange curves (second) are the training loss for controller model $\hat{\pi}_\phi$. Green curves (third) are the training loss for robust controller model with noise injection $\hat{\pi}'_\phi$. The red downward arrows point out the loss drop because of the learning rate decay.

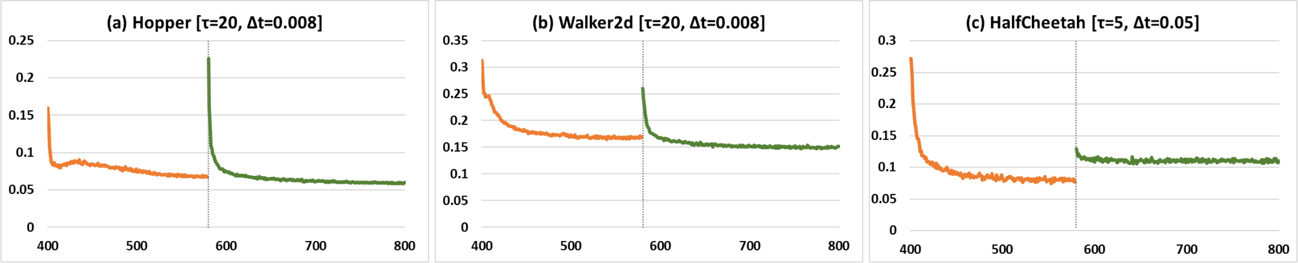

Figure 6. Imitation loss versus training epochs. The x-axis is the number of training epochs. The y-axis is the imitation loss computed by the L2 norm of the difference between the true action inputs $u_{e,t_{i:(i+\tau)}}$ and the predicted action inputs $\hat{\boldsymbol{u}}_{t_{i:(i+\tau)}}$. Orange curves (first) are the imitation loss with respect to the controller training procedure $\mathcal{L}_\phi(\tau)$. Green curves (second) are the imitation loss with respect to the robustness enhancement procedure $\mathcal{L}'_\phi(\tau)$.

C.1. Training Loss and Imitation Loss

Training loss

As described by Algorithm.1 in the main essay, we could observe the three training phases inside the dynamics-controller module from Fig.5, where the sudden loss jumps between the blue curve and the orange curve represents the switching from the dynamics training to the controller training via freezing the learned dynamics network $\hat{f}_\theta$ and enabling the initialized controller network $\hat{\pi}_\phi$. On the other hand, the jump between the orange curve and the green curve represents the beginning of the controller robustness enhancement $\hat{\pi}'_\phi$ procedure via noise injection.

Imitation loss

Instead of the BC-like methods which determine the training performance based on the imitation loss ($\| \boldsymbol{u}_{e,t_{i:(i+\tau)}} - \hat{\boldsymbol{u}}_{t_{i:(i+\tau)}} \|^2$), the proposed RMBIL framework is determined by the state loss ($\| \boldsymbol{x}_{e,t_{i:(i+\tau)}} - \hat{\boldsymbol{x}}'_{t_{i:(i+\tau)}} \|^2$) as shown in Fig.5. Therefore, in order to evaluate the imitation loss of the proposed framework, we make the trained controller ($\hat{\pi}_\phi$ and $\hat{\pi}'_\phi$) predict a sequence of action inputs $\hat{\boldsymbol{u}}_{t_{i:(i+\tau)}}$ with a certain horizon τ (same as the value for computing the state loss $\mathcal{L}(\tau)$) at each training epoch. The results are shown in Fig.6, where the loss jump represents the beginning of the noise injection process.

C.2. Implementation Details

Demonstrations

The performance of the expert and random demonstrations for the selected three environments are listed in Table.2, where mean and standard deviation are computed based on 50 test episodes with 1000 steps. In addition, in the robustness evaluation experiments, we neglect the termination signal received from the OpenAI gym interface since the imitated policies may drive the fallen down system to stand up and move forward again. In other words, we accumulate the reward signal continuously even the system falls down.

Table 2. Performance of demonstrations.

Environment	Observation space	Action space	Expert policy	Random policy
Hopper-v2	11 (continuous)	3 (continuous)	3779.57±3.36	919.94±104.02
Walker2d-v2	17 (continuous)	6 (continuous)	5520.24±53.00	966.00±149.56
HalfCheetah-v2	17 (continuous)	6 (continuous)	4219.44±87.37	-284.94±69.79

The reason for selecting these three environments to evaluate the proposed RMBIL is because we are interested in the complex dynamics with collision and unstable locomotion. In the default OpenAI gym environments under Mujoco physical engine, there are five tasks meet the requirements: Hopper, Walker, HalfCheetah, Ant, and Humanoid. However, we exclude the Ant-v2 and Humanoid-v2 since their observation space contains the external force and collision information, which can not be treated as the system state. Otherwise, the assumptions for training a NDI controller via Neural ODE would not hold.

Hyperparameters

Although the Algorithm.1 in the main essay states that the three training phases inside the dynamics-controller module will be complete if the training loss is lower than the corresponding user-specified value (ϵ), it is hard to find a proper value in practice since the optimal value varies from case to case (depending on the selected environment and hyperparameter settings). Fortunately, we observed that such training procedures always converge under enough training epochs (shown in Fig.5). Therefore, in Table.3, we provide a reference of hyperparameters setting for training both dynamics-control and generator modules depending on the number of epochs. The suggested reference setting is suitable for common OpenAI control tasks, such as Pendulum, Hopper, Walker2d, etc.

Table 3. Reference settings for hyperparameters.

Hyperparameters for the dynamics-control module (Neural ODE)					
Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
number of hidden neuron (θ)	800	number of hidden layers (θ)	2	activation of the hidden layers (θ)	ELU
number of hidden neuron (ϕ)	320	number of hidden layers (ϕ)	2	activation of the hidden layers (ϕ)	ELU
number of epochs for f_θ	400	init. learning rate for f_θ	0.01	learning rate decay	0.5 per 100 epochs
number of epochs for π_ϕ	180	init. learning rate for π_ϕ	0.001	noise standard deviation $\sigma_{\boldsymbol{x}}$	0.25
number of epochs for π'_ϕ	220	batch size	2048	absolute tolerance for ode-solver	1e-4
type of ode-solver	adams	prediction horizon τ	4∼25	relative tolerance for ode-solver	1e-4
Hyperparameters for the generator module (CVAE)					
Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
number of hidden neuron (ψ)	320	number of hidden layers (ψ)	2	activation of the hidden layers (ψ)	ELU
number of epochs for ψ_e, ψ_d	1000	latent dimension $\boldsymbol{z}$	5∼10	learning rate	0.001

where the selected adams method is an ode-solver with adaptive step size. Compared to the fixed-step method, the variable-step method could stay convergent when the system becomes stiff (numerically unstable) and get better accuracy. Unfortunately, the variable-step method may suffer from the extremely long integration time on an unstable system (due to extremely small step), which is the major cause of the inference time bottleneck for the learned dynamics. However, the proposed RMBIL framework does not rely on the learned dynamics at inference, therefore, the learned robust NDI controller would benefit from the accuracy provided by the variable-step method but also get rid of the inference time bottleneck issue. In general, the suggested values in Table.3 could directly apply to an imitation learning task and get a converged result. The only parameter needs to be tuned manually is the prediction horizon τ. Empirically, we let the product of τ and sampling time step (which is equivalent to the integration time step Δt) to be smaller than 0.2 sec long in order to take the balance between prediction accuracy and training time. Take Hopper-v2 as an example, since its environment time step (Δt) is 0.008, the suitable value for the prediction horizon τ should be smaller than 0.2/0.008 = 25 steps.

Network structure

As mentioned in the main essay, only the decoder ψ_d of the trained CVAE and learned robust NDI controller π'_ϕ are used for imitating expert behavior at inference. Therefore, for a fair comparison, we force the network structure of the imitated policy for each baseline, that is BC, GAIL and DART, should contain 4 hidden layers with 320 neurons and ELU activation function. The reason for choosing ELU as the activation function is because non-smooth non-linearities functions such as ReLU and LeakyReLU may cause non-unique solutions during the backpropagation via the adjoint method (Chen et al., 2018). In addition, in our preliminary test, ELU could obtain the best prediction accuracy among common smooth non-linearities functions with theoretically unique gradients, such as Softplus and Tanh.

C.3. Uneven Environments

The screenshots for Hopper and Walker2d tasks with uneven surface are illustrated in Fig.7, where the 'UnevenEnv' for Walker2d task (Fig.7-(a)) is built with a series of 2m span boxes with random heights (0∼20**mm**). Hopper-UnevenEnv (Fig.7-(c)) has similar settings but with 1m span boxes because the walking distance for Hopper is much shorter than Walker2d. In contrast, the settings of the 'SlopeEnv', which has two cases of tilted angle, are the same for both tasks

(Fig.7-(b) and (d)). In addition, the friction for both types of uneven surface is identical to the default flat surface.

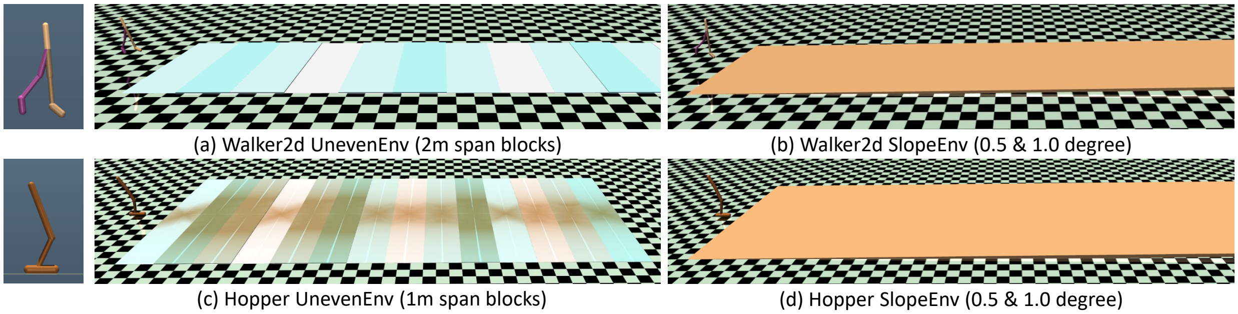
(a) Walker2d UnevenEnv (2m span blocks)
(b) Walker2d SlopeEnv (0.5 & 1.0 degree)
(c) Hopper UnevenEnv (1m span blocks)
(d) Hopper SlopeEnv (0.5 & 1.0 degree)

Figure 7. Screenshots for uneven environments used in robustness evaluation experiments.

D. Neural ODE Based Multi-Steps Actuated Dynamics

Fig.8 shows the architecture illustration of the proposed Neural ODE based multi-steps continuous actuated dynamics. In addition, this section provides experimental evaluations of the proposed Neural ODE dynamics during training and inference. Sec.D.1 describes the setup for the trajectories prediction experiments and Sec.D.2 shows the experimental results compared to the baseline. In addition, Sec.D.3 shows the open-loop trajectory state diagnostics compared to the expert demonstration.

D.1. Experiments Setup

Neural ODE dynamics

In order to demonstrate that the learned neural network parameterized by θ inside the proposed RMBIL framework is in fact a precise multi-steps actuated dynamics, which is the key assumption of Proposition 3.1 in the main essay, we bypass the trained robust controller $\hat{\pi}'_\phi$ inside the dynamics-controller module used in Sec.C. The remaining activated network model $\hat{f}_\theta$ along with ode-solver is equivalent to a Neural ODE based actuated dynamics using ZOH method for handling the external action inputs. Therefore, given an initial state ($\boldsymbol{x}_{t_0}$), a series of integration time steps ($t_0, ..., t_n$) and corresponding action inputs sequence ($\boldsymbol{u}_{t_{0:n}}$), the learned Neural ODE dynamics could predict a trajectory rollout for n steps ($\hat{\boldsymbol{x}}_{t_{0:n}}$).

Baselines

To compare the proposed multi-steps continuous Neural ODE dynamics to a typical one-step forward discrete dynamics, which can be formulated as $\hat{\boldsymbol{x}}_{(i+1)} = \hat{f}_\omega(\boldsymbol{x}_i, \boldsymbol{u}_i)$, we use a two layers MLP (with the same neuron number and activation functions as the model $\hat{f}_\theta$) as our baseline. Then, the defined MLP model, parameterized by ω, is trained with the same expert demonstrations (Sec.C.2) under the same state loss function but the prediction horizon is one step forward ($\mathcal{L}_\omega(\tau = 1)$).

D.2. Performance Evaluation

Accumulated error comparison

The long term accuracy of the learned dynamics is the key assumption for training a NDI controller via Neural ODE (Proposition 3.1), therefore, to effectively evaluate the performance of the multi-steps (n) trajectory prediction, we define the evaluation loss ($\mathcal{L}_{\text{eval}}(s)$) by accumulating the error between the predicted state ($\hat{\boldsymbol{x}}_{t_i}$) and true state ($\boldsymbol{x}_{e,t_i}$, which is the expert demonstration) at each prediction step (s), as follows:

$$\mathcal{L}_{\text{eval}}(s) = \frac{1}{n}\sum_{i=0}^{s} \parallel \hat{\boldsymbol{x}}_{t_i} - \boldsymbol{x}_{e,t_i} \parallel^2 , \text{ where } 0 \leq s \leq n. \tag{35}$$

To compare the evaluation loss between the proposed Neural ODE dynamics ($\hat{f}_\theta$) and the baseline ($\hat{f}_\omega$), we make inferences on both trained models to predict n steps trajectories by giving the same initial state ($\boldsymbol{x}_0$) and a sequence of action inputs ($\boldsymbol{u}_{t_{0:n}}$). The result is illustrated in Fig.9, where both Hopper and Walker2d tasks are evaluated for 200 steps (total duration is 200x0.008=1.6 seconds long) while HalfCheetah is evaluated for 50 steps (total duration is 50x0.05=2.5 seconds long). We could observe that Neural ODE dynamics outperforms the baseline as the prediction steps increases. For tasks with small sampling time step, such as Hopper and Walker2d ($\Delta t = 0.008$), the divergence rate of accumulated error for Neural ODE dynamics is relatively stable than the task with large sampling time step such as HalfCheetah ($\Delta t = 0.05$). This phenomenon fits the characteristics of the ode-solver, that is, the integration error decreases as the time step shrinking.

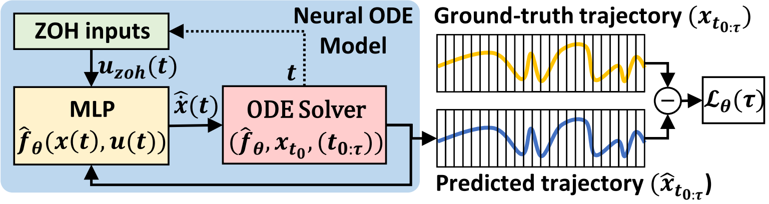
ZOH inputs
Neural ODE Model
$u_{zoh}(t)$
t
MLP
$\hat{f}_\theta(x(t), u(t))$
$\hat{x}(t)$
ODE Solver
$(\hat{f}_\theta, x_{t_0}, (t_{0:\tau}))$
Ground-truth trajectory $(x_{t_{0:\tau}})$
$-$
$\mathcal{L}_\theta(\tau)$
Predicted trajectory $(\hat{x}_{t_{0:\tau}})$

Figure 8. Actuated Neural ODE Model. Yellow block is a neural network for computing the time derivative of actuated dynamics. Light green block is a continuous function $\boldsymbol{u}_{\text{zoh}}(t)$ approximated by ZOH method based on discrete input sequence $\boldsymbol{u}_{e,t_i}$. Although the internal state $\boldsymbol{x}(t)$ is continuous, the predicted outputs are discrete with regard to the specified time sequence $t_{0:\tau}$. Hence, the loss can be computed based on both discrete trajectories $\boldsymbol{x}_{t_{0:\tau}}, \hat{\boldsymbol{x}}_{t_{0:\tau}}$.

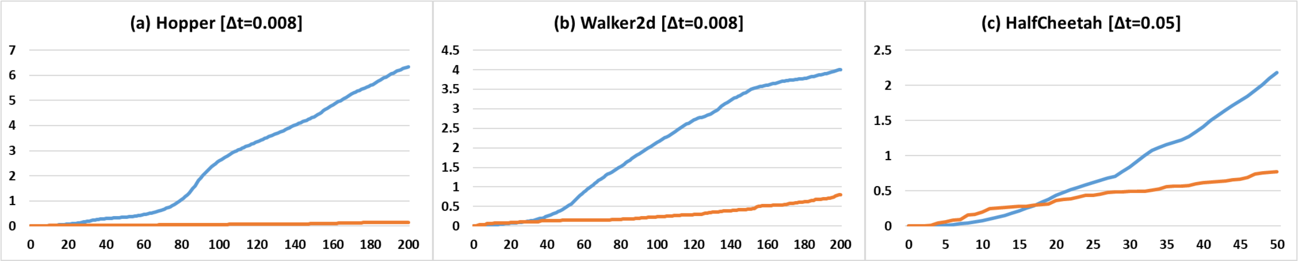

Figure 9. Trajectory accumulated error comparison between the proposed Neural ODE dynamics and the baseline. The x-axis is the forward prediction step. The y-axis is the accumulated errors with respect to the expert demonstrations (Eq.35). Blue curves are trajectories predicted by the baseline ($\hat{f}_\omega$). Orange curves are trajectories predicted by the Neural ODE dynamics ($\hat{f}_\theta$).

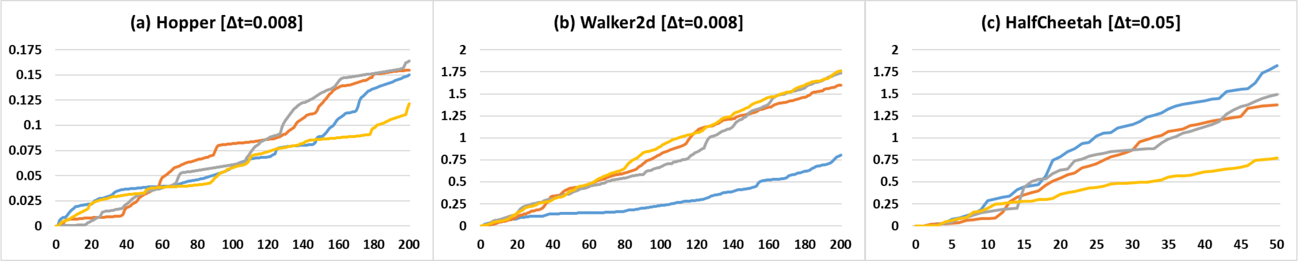

Figure 10. Trajectory accumulated error comparison for different initial states using Neural ODE dynamics. The x-axis is the forward prediction step. The y-axis is the accumulated errors with respect to the expert demonstrations (Eq.35). The different color curve represent trajectory prediction started from the different initial state (randomly selected).

Random initial state analysis

The accumulated error comparison evaluated in the previous part is based on the specific initial state (which is exactly the first state from the expert demonstrations $\boldsymbol{x}_{e,t_0}$), however, the imitated controller $\hat{f}_\theta$ may start from any state among the demonstrations at inference. Therefore, we are interested in how the initial state, randomly selected from the demonstrations, affects the performance of trajectory prediction. Fig.10 shows the trajectory predictions comparison for four different initial states. We could observe that the performance of accumulated errors across different initial states on low-dimensional systems, such as Hopper, is more consistent than the high-dimensional systems, such as Walker2d or HalfCheetah.

D.3. Open-Loop State Diagnostics

The following three figures demonstrate that the learned Neural ODE dynamics ($\hat{f}_\theta$) could precisely reproduce the expert demonstrations under the open-loop forward prediction given the initial state ($\boldsymbol{x}_{e,t_0}$) and a sequence of action inputs ($\boldsymbol{u}_{e,t_{0:n}}$). As mentioned in the main essay, the accurate multi-steps trajectory prediction is the core of the proposed RMBIL framework.

Hopper task

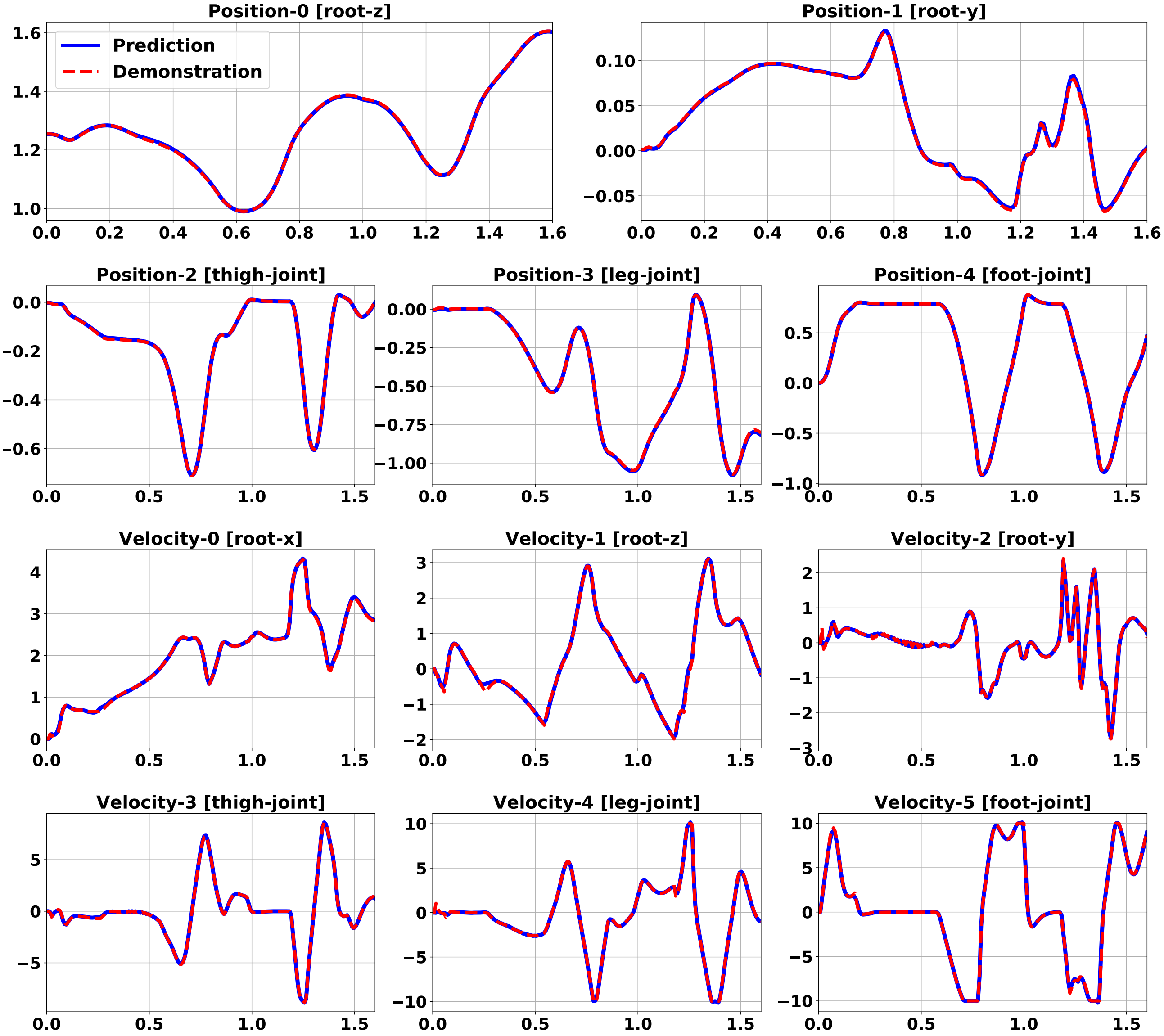

Figure 11. The open-loop trajectory prediction state diagnostics on Hopper task. The x-axis is the time step (in sec). The y-axis is the state value. The red dash lines are ground truth trajectory from the expert demonstration. The solid blue lines are the multi-steps trajectory (200 steps) predicted by the learned Neural ODE dynamics ($\hat{f}_\theta$) given the initial state ($\boldsymbol{x}_{t_0}$) and a sequence of action inputs ($\boldsymbol{u}_{e,t_{0:200}}$).

Walker2d task

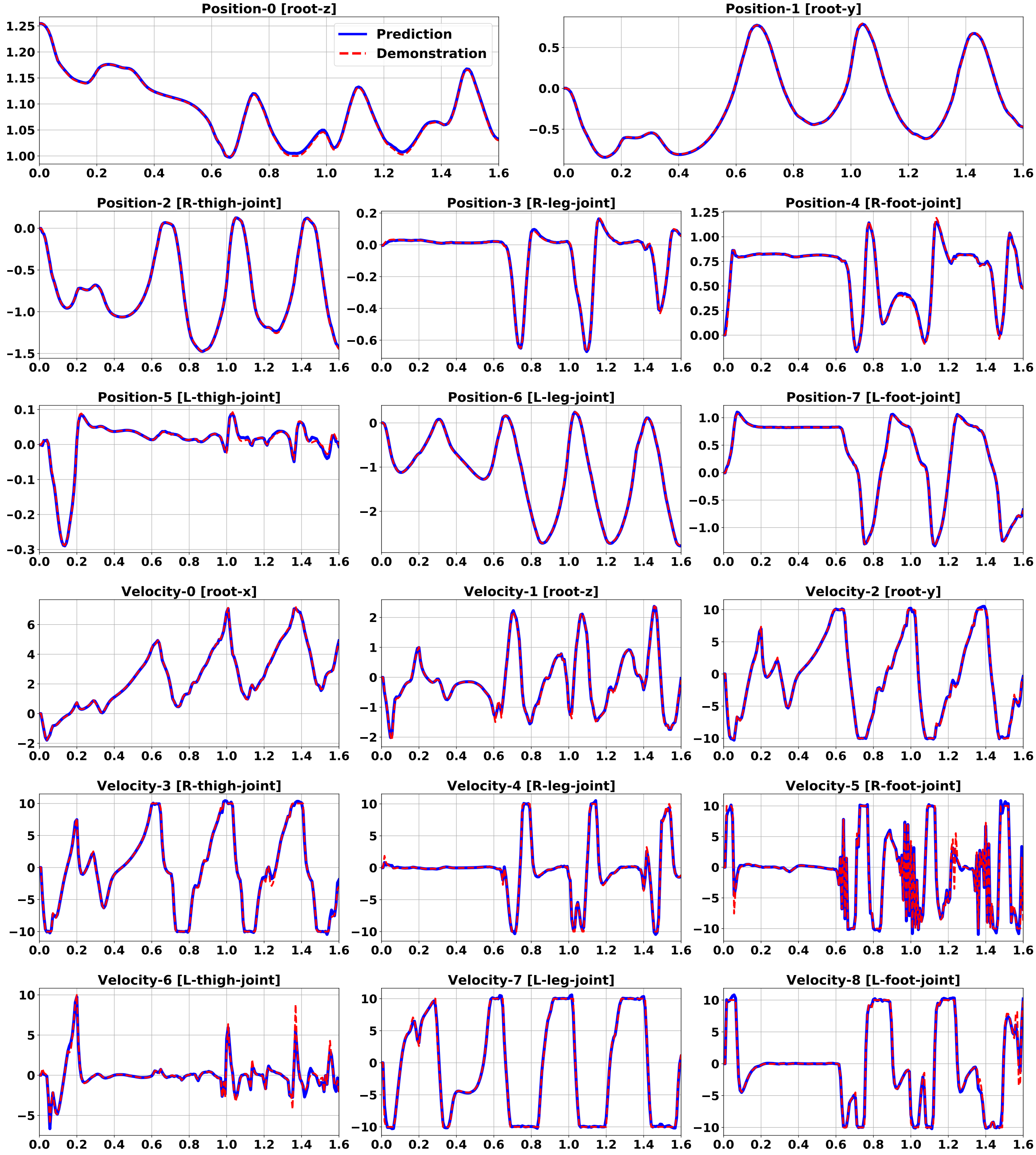

Figure 12. The open-loop trajectory prediction state diagnostics on Walker2d task. The x-axis is the time step (in sec). The y-axis is the state value. The red dash lines are ground truth trajectory from the expert demonstration. (The dense red lines area in the Velocity-5 subplot is caused by high-frequency motion on the right foot joint.) The solid blue lines are the multi-steps trajectory (200 steps) predicted by the learned Neural ODE dynamics ($\hat{f}_\theta$) given the initial state ($\boldsymbol{x}_{t_0}$) and a sequence of action inputs ($\boldsymbol{u}_{e,t_{0:200}}$).

HalfCheetah task

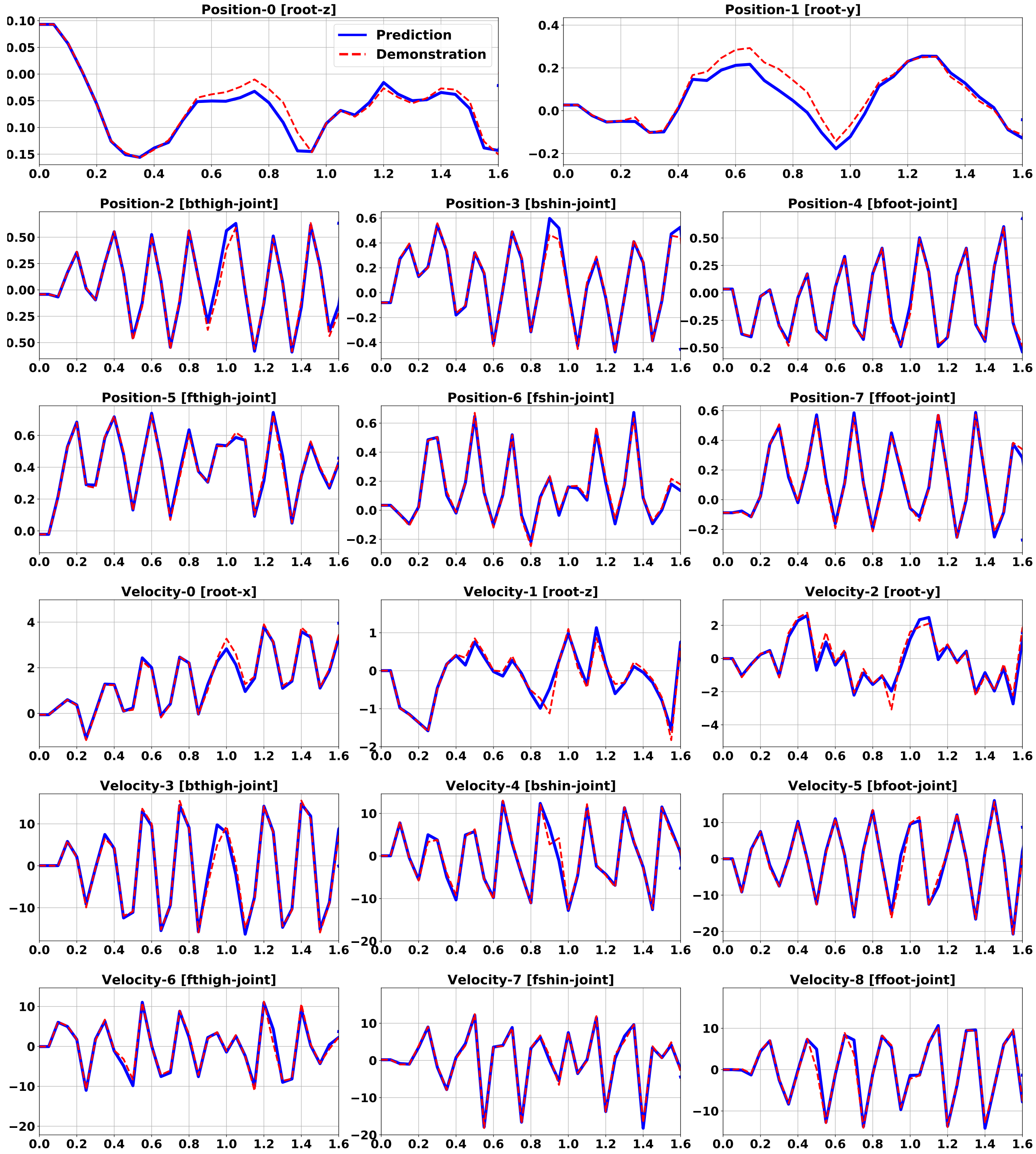

Figure 13. The open-loop trajectory prediction state diagnostics on HalfCheetah task. The x-axis is the time step (in sec). The y-axis is the state value. The red dash lines are ground truth trajectory from the expert demonstration. The solid blue lines are the multi-steps trajectory (50 steps) predicted by the learned Neural ODE dynamics ($\hat{f}_\theta$) given the initial state ($\boldsymbol{x}_{t_0}$) and a sequence of action inputs ($\boldsymbol{u}_{e,t_{0:50}}$).