

Neural Robot Dynamics

Appendix

Submission ID: 222

Contents

A.1 Additional <i>NeRD</i> Details	2
A.2 Additional Training Details and Hyperparameters	3
A.3 Additional Experiment Details	3
A.3.1 Details of Policy Learning Tasks	3
A.3.1.1 Cartpole	3
A.3.1.2 Ant	4
A.3.1.3 Franka	4
A.3.1.4 ANYmal	5
A.3.2 Visualization and Full Results for Double Pendulum with Varying Contact Environments	5
A.3.3 Ablation Study	5
A.3.3.1 Network Architecture	6
A.3.3.2 Hybrid Prediction Framework	6
A.3.3.3 Relative Robot State Prediction	7
A.3.3.4 Robot-Centric State Representation	7
A.3.3.5 Model Input and Output Normalization	7
A.3.3.6 History Window Size h	7

A.1 Additional *NeRD* Details

We provide additional details about *NeRD* that are not covered in the main paper due to space constraints.

Contact-Related Quantities We use contact-related quantities \mathcal{C}_t as an application-agnostic representation to capture how the surroundings impact the robot’s dynamics, without the need to parameterize the whole environment. This is inspired by the dynamics and contact solvers of analytical simulators, as these solvers also use contact-related quantities to formulate physics equations to evolve the robot dynamics. We construct $\mathcal{C}_t = \{\mathbf{c}_t^i\}$ by reusing the collision detection module in the classical simulator. For each pre-specified contact point \mathbf{p}_0^i on the robot, we obtain its contact event quantities $\mathbf{c}_t^i = (\mathbf{p}_0^i, \mathbf{p}_1^i, \vec{\mathbf{n}}^i, d^i)$. Here \mathbf{p}_1^i is the contact point on a non-robot shape, $\vec{\mathbf{n}}^i$ is the contact normal, and d^i is the contact distance (zero or negative for collisions). We mask a \mathbf{c}_t^i to be zero if the associated contact distance is larger than a positive threshold $d^i > \xi$, allowing free-space motion while providing robustness to cases where a collision occurs within the timestep. Quantity ξ is a positive value greater than or equal to the *contact thickness* of a geometry (*i.e.*, a standard collision-detection parameter in classical simulators). Ideally, ξ should also exceed the allowed maximum displacement of the contact point within a timestep, ensuring that all near-contact events are retained. However, in practice, the choice of ξ is very flexible, and we use a fixed setting of $\xi = \max(4 \cdot \text{contact_thickness}, 0.1)$ across all our experiments without task-specific tuning.

Robot-Centric State Representation Here we provide the detailed calculation for transforming the robot state into robot’s base frame. For the robot state \mathbf{s}_k at time step $k \in [t - h + 1, t]$, we transform it into robot’s base frame, \mathbf{B}_k , at time step k , where $\mathbf{B}_k = (\mathbf{x}_k, \mathbf{R}_k)$. For the robot articulation, we use the reduced coordinate state, which is spatially invariant; thus, we only need to transform the state of the robot base (*i.e.*, $\mathbf{x}_k, \mathbf{R}_k$, and ϕ_k) into the robot’s base frame. Therefore, we have $\mathbf{s}_k^{\mathbf{B}_k} = (\mathbf{x}_k^{\mathbf{B}_k}, \mathbf{R}_k^{\mathbf{B}_k}, \mathbf{q}_k, \phi_k^{\mathbf{B}_k}, \dot{\mathbf{q}}_k)$, with

$$\mathbf{x}_k^{\mathbf{B}_k} = \mathbf{0}, \quad (1)$$

$$\mathbf{R}_k^{\mathbf{B}_k} = \text{Identity}, \quad (2)$$

$$\boldsymbol{\nu}_k^{\mathbf{B}_k} = \mathbf{R}_k^{-1}(\boldsymbol{\nu}_k - \mathbf{x}_k \times \boldsymbol{\omega}_k), \quad (3)$$

$$\boldsymbol{\omega}_k^{\mathbf{B}_k} = \mathbf{R}_k^{-1}\boldsymbol{\omega}_k, \quad (4)$$

where $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ are the linear and angular components of a spatial twist ϕ , respectively.

Additionally, the predicted state difference $\Delta \mathbf{s}_{t+1}$ (*i.e.*, the network’s output) is expressed in the robot’s base frame at time step t instead of $t + 1$, *i.e.*, $\Delta \mathbf{s}_{t+1}^{\mathbf{B}_t} \triangleq \mathbf{s}_{t+1}^{\mathbf{B}_t} \ominus \mathbf{s}_t^{\mathbf{B}_t}$. This is because, when expressed in its own base frame, the state of the robot base (\mathbf{x}, \mathbf{R}) is always the identity transformation (*i.e.*, $\mathbf{x}_t^{\mathbf{B}_t} = \mathbf{x}_{t+1}^{\mathbf{B}_t} = \mathbf{0}$ and $\mathbf{R}_t^{\mathbf{B}_t} = \mathbf{R}_{t+1}^{\mathbf{B}_t} = \mathbf{I}$), which results in zero state changes for these dimensions, so $\Delta \mathbf{x}_{t+1}^{\mathbf{B}_t} = \mathbf{0}$ and $\Delta \mathbf{R}_{t+1}^{\mathbf{B}_t} = \mathbf{0}$. Therefore the learned model cannot predict the motion of the robot base if using $\Delta \mathbf{s}_{t+1}^{\mathbf{B}_{t+1}}$ as the prediction target.

To compute $\mathbf{s}_{t+1}^{\mathbf{B}_t} = (\mathbf{x}_{t+1}^{\mathbf{B}_t}, \mathbf{R}_{t+1}^{\mathbf{B}_t}, \mathbf{q}_{t+1}, \phi_{t+1}^{\mathbf{B}_t}, \dot{\mathbf{q}}_{t+1})$, we use the following calculations:

$$\mathbf{x}_{t+1}^{\mathbf{B}_t} = \mathbf{R}_t^{-1}(\mathbf{x}_{t+1} - \mathbf{x}_t), \quad (5)$$

$$\mathbf{R}_{t+1}^{\mathbf{B}_t} = \mathbf{R}_t^{-1}\mathbf{R}_{t+1}, \quad (6)$$

$$\boldsymbol{\nu}_{t+1}^{\mathbf{B}_t} = \mathbf{R}_t^{-1}(\boldsymbol{\nu}_{t+1} - \mathbf{x}_t \times \boldsymbol{\omega}_{t+1}), \quad (7)$$

$$\boldsymbol{\omega}_{t+1}^{\mathbf{B}_t} = \mathbf{R}_t^{-1}\boldsymbol{\omega}_{t+1}, \quad (8)$$

Multi-Substep Prediction To improve the stability of the simulation, a classical simulator often utilizes a smaller timestep (*i.e.*, substep) and runs multiple substeps of solver-integration iterations to obtain the actual state of the robot at the next time step, \mathbf{s}_{t+1} (as shown in Fig. 2(a)). Unlike previous neural simulation works [1, 2, 3] that sequentially predict the robot state acceleration at each substep and obtain the robot state at next time step by time integration over substeps, *NeRD*

directly predicts the state difference from the current robot state to the state at the next (macro) time step, s_{t+1} , which might span multiple substeps in the analytical simulator. This design enables us to learn *NeRD* from a finer-grained simulator with smaller substep sizes without sacrificing the efficiency of the learned model at test time.

A.2 Additional Training Details and Hyperparameters

We adopt a lightweight implementation of causal Transformer [4, 5], and repurpose it as a sequential model for robot dynamics. Past robot-centric simulation states are encoded into embeddings via a learnable linear layer, processed through Transformer blocks with self-attention, and then mapped to latent features from which the robot state difference is predicted as the output.

We use a fixed set of hyperparameters for training *NeRD* across all six robotic systems – including training hyperparameters and Transformer hyperparameters – except for the embedding size of the Transformer model. For robots with fewer degrees of freedom, we use a smaller embedding size to enhance training and inference efficiency. The complete hyperparameter settings used in our experiments are provided in Table A.1.

Table A.1: Training hyperparameters in the experiments.

Robot		Cartpole	Pendulum	Cube Tossing	Franka	Ant	ANYmal
Training	history window size h	10					
	batch size	512					
	learning rate	linear decay from $1e^{-3}$ to $1e^{-4}$					
Transformer	block size	32					
	num layers	6					
	num heads	12					
	input embedding size	192			384		
	dropout	0					
Output MLP	num layers	1					
	layer size	64					

A.3 Additional Experiment Details

A.3.1 Details of Policy Learning Tasks

We train a *NeRD* model for each robotic system and use the trained *NeRD* model in all the downstream tasks for the corresponding robotic system. We provide details of the policy learning tasks demonstrated in §5.3.

A.3.1.1 Cartpole

Swing-Up Task In this task, a *Cartpole* (2-DoF) is controlled to swing up its pole from a randomized initial angle to be upright and maintain the upright pose as long as possible. The action space is joint-space torque, and the *Cartpole* is directly controlled by the commanded joint-space torque. The observation of the policy is 4-dimensional, including:

- 2-dim joint positions: x, θ
- 2-dim joint velocities: $\dot{x}, \dot{\theta}$

A trajectory is terminated if it exceeds the maximum number of steps 300, or the cart position of the *Cartpole* moves outside the $[-4\text{ m}, 4\text{ m}]$ interval, or the joint velocity is above 10 rad/s .

The stepwise reward function is below:

$$\mathcal{R}_t = 5 - \theta_t^2 - 0.05x_t^2 - 0.1\dot{\theta}_t^2 - 0.1\dot{x}_t^2.$$

A.3.1.2 Ant

Ant Running In this task, an *Ant* robot (14-DoF) is controlled to move forward as fast as possible. The action space is joint-space torque, and the *Ant* is directly controlled by the commanded joint-space torque. The observation space has 29 dimensions, including:

- 1-dim height of the base h
- 4-dim orientation of the base represented by a quaternion
- 3-dim linear velocity of the base v
- 3-dim angular velocity of the base ω
- 8-dim joint positions
- 8-dim joint velocities
- 2-dim up and heading vector projections: $p_{\text{up}}, p_{\text{heading}}$.

The episode is terminated if it exceeds the maximum number of steps 500, or the height of the base reaches $h < 0.3 \text{ m}$.

The stepwise reward function for the running task is defined below:

$$\mathcal{R}_t = v_x + 0.1p_{\text{up}} + p_{\text{heading}}.$$

Ant Spinning An *Ant* is controlled to maximize its spinning speed around the gravity axis (Y-axis in this environment) in this task. It uses the same observation space and the termination condition as the running task. The stepwise reward function is defined as below:

$$\mathcal{R}_t = \omega_y + p_{\text{up}}.$$

Ant Spin Tracking This task requires an *Ant* to track a spinning speed of 5 rad/s . It uses the same observation space and termination condition as the running task. The stepwise reward function is defined below:

$$\mathcal{R}_t = 5 \cdot \exp(-(\omega_y - 5)^2 - 0.1\omega_x^2 - 0.1\omega_z^2) + 0.1p_{\text{up}}.$$

A.3.1.3 Franka

End-Effector Reach Task The goal of this task is to move the Franka robot’s end-effector to a randomly-specified target position. The action space is defined as delta joint positions, which are executed via a *joint-position PD controller* (Note: the *NeRD* model is still predicting using the joint-space torques, which are converted from the target joint positions via the joint-position PD controller). The 13-dim observation space consists of the following:

- 7-dim joint positions
- 3-dim end-effector position
- 3-dim target goal position

The episode length of this task is 128. We adopt an exponential reward function from our prior works, with minimal tuning:

$$\mathcal{R}_t = -d + \frac{1}{\exp(50d) + \exp(-50d) + \epsilon} + \frac{1}{\exp(300d) + \exp(-300d) + \epsilon},$$

where $d = \|\vec{e}\|$ is the end-effector’s distance to the goal position and $\epsilon = 0.0001$.

A.3.1.4 ANYmal

Forward Walk Velocity-Tracking In this task, an *ANYmal* robot [6] (18-DoF) is controlled to track a forward walking speed of 1 *m/s*. The action space is the target joint positions, and the *ANYmal* is controlled by a *joint-position PD controller*. The observation space is similar to *Ant* tasks and has 37 dimensions, including the following:

- 1-dim height of the base h
- 4-dim orientation of the base represented by a quaternion
- 3-dim linear velocity of the base \mathbf{v}
- 3-dim angular velocity of the base $\boldsymbol{\omega}$
- 12-dim joint positions
- 12-dim joint velocities
- 2-dim up and heading vector projections: $p_{\text{up}}, p_{\text{heading}}$.

The episode is terminated if it exceeds the maximum number of steps 1000, or the height of the base $h < 0.4$ *m*, or the base or the knees hit the ground. We adopt a commonly used reward function [7] with minimal tuning:

$$\mathcal{R}_t = \exp\left(-((\mathbf{v}_x - 1)^2 + \mathbf{v}_z^2)\right) + 0.5 \exp(-\boldsymbol{\omega}_y^2) - (0.002 \sum \boldsymbol{\tau})^2,$$

where $\boldsymbol{\tau}$ is the joint-space torque. Note that, in our *ANYmal* environments, \vec{x} is the forward direction, \vec{z} is the sideways direction, and \vec{y} is the upward direction.

Sideways Walk Velocity-Tracking This task requires an *ANYmal* robot to track a sideways walking speed of 1 *m/s*. It uses the same action space, observation space, and termination condition as the *Forward Walk Velocity-Tracking task*. The reward function is below:

$$\mathcal{R}_t = \exp\left(-(\mathbf{v}_x^2 + (\mathbf{v}_z - 1)^2)\right) + 0.5 \exp(-\boldsymbol{\omega}_y^2) - (0.002 \sum \boldsymbol{\tau})^2.$$

A.3.2 Visualization and Full Results for Double Pendulum with Varying Contact Environments

We provide visualizations of the seven ground configurations and the detailed measured errors in this section. The seven contact setups include one contact-free scenario where we put the ground far below the pendulum, and six different planar ground settings where the double pendulum is able to make contact with the ground (as shown in Fig. A.1). We use a single trained *NeRD* model for all contact setups and generate 2048 passive-motion trajectories of the *Double Pendulum* over a duration of 100 steps with random initial states. We report the mean joint-angle errors (in *radians*) over the trajectory in Table A.2.



Figure A.1: Seven *Double Pendulum* contact configurations used for testing *NeRD*'s generalizability across different contact environments.

A.3.3 Ablation Study

The success of *NeRD* relies on several critical design decisions made during development. In this section, we analyze these design decisions through a series of ablation experiments. Specifically,

Table A.2: **Full passive-motion evaluation results on *Double Pendulum*.** For each contact configuration, we report the mean joint-angle error of each joint in radians.

Robot	Double Pendulum						
Contact Configuration	no contact	ground #1	ground #2	ground #3	ground #4	ground #5	ground #6
Joint #1 Error (<i>rad</i>)	0.004	0.012	0.008	0.005	0.011	0.029	0.008
Joint #2 Error (<i>rad</i>)	0.007	0.015	0.011	0.011	0.018	0.056	0.013

we conduct our study using two evaluation test cases: (1) contact-free passive motion of *Double Pendulum*; and (2) policy evaluation on the *Ant* running task. All ablation models are trained on the same datasets as their corresponding *NeRD* models.

For test case #1, we compute the temporally-averaged mean joint-angle error (average error of two joints) of *Double Pendulum* for each ablation model, from 2048 passive motion trajectories of the *Double Pendulum* over a duration of 100 steps with random initial states. Then we normalize the errors by the error of the *NeRD* model, and report the values in Fig. A.2 (first row).

For test case #2, we execute the three *Ant* running policies trained in our policy-learning experiments and evaluate the average reward obtained in each ablation neural dynamics model (§5.3) (2048 trajectories for each policy in each ablation neural dynamics model). For each ablation model, we then compute the reward differences compared to the reward obtained in the ground-truth simulator. The reward differences are then normalized by the reward difference of the *NeRD* model and reported in Fig. A.2 (second row).

A.3.3.1 Network Architecture

During development, we found the Transformer architecture to be the most effective for modeling neural robot dynamics. We demonstrate this by comparing it against three other architectures in Fig. A.2(a): **MLP**: a baseline model that predicts state changes from the robot-centric simulation state of the current step; **GRU** and **LSTM**: two RNN architectures that leverage historical state information in their predictions. Although the ground-truth simulator computes the dynamics in a stateless way (*i.e.*, the next state only depends on the current state and torques), we found that sequence modeling is important to achieve high accuracy of the neural robot dynamics model. We hypothesize that the high variance of the velocity inputs is a challenge for the model; by including the historical states as input, the neural model is able to infer a smoothed version of velocity and combine it with the actual velocity input for a better prediction. Based on the comparisons for policy evaluation on the *Ant* running task, the causal Transformer model helps achieve a reward that is much closer to the ground-truth simulator, compared to RNN architectures.

A.3.3.2 Hybrid Prediction Framework

To demonstrate the effectiveness of our *Hybrid Prediction Framework*, we compare *NeRD* against an *End-to-End* prediction baseline (**E2E**), which directly maps robot state and action to the next robot state. This *end-to-end* framework is commonly adopted by prior neural simulators for rigid bodies [8, 9, 10]. We reimplemented it in the Warp simulator. Specifically, in our implementation, the **E2E** baseline maps the robot state and the joint torques to the next robot state, and the robot state is expressed in the world frame. We replace the action input commonly used in *End-to-End* approaches with joint-torque input so that we can use the same training dataset as *NeRD* for a fair comparison. Fig. A.2(b) shows that the **E2E** baseline has large prediction errors in both test cases. This is because, in the *Double Pendulum* case, the training dataset consists of varying scenarios of contact configurations. However, the **E2E** state representation without encoding the environment provides insufficient clues to differentiate distinct contact configurations during training, thus resulting in poor performance. In the *Ant* test case, the **E2E** baseline fails because the world-frame robot state cannot make a reliable prediction when the *Ant* moves far away from the origin and reaches regions outside the range of the training dataset.

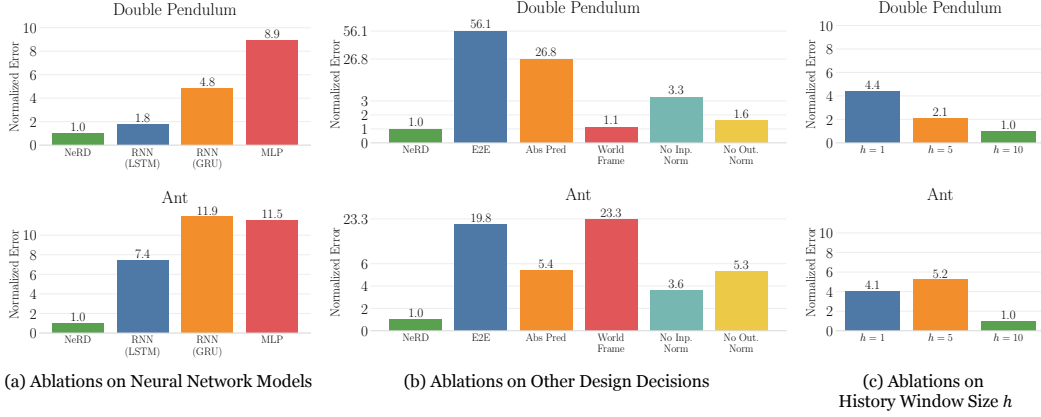


Figure A.2: **Ablation Study.** We evaluate ablation variants on two test cases: contact-free passive motion of the *Double Pendulum* and policy evaluation on the *Ant* running task. We normalize the errors by the error of *NeRD* ($h = 10$). (a) Ablations of different neural network architectures; (b) Ablations of other critical design decisions in *NeRD*. (c) Ablations on the history window size h .

A.3.3.3 Relative Robot State Prediction

The third key design decision is the use of relative robot state prediction, where the model predicts the state difference between the current robot state and the robot state in the next time step, rather than directly predicting the absolute next robot state. Predicting the relative state changes effectively reduces the range of values of the model output, thus stabilizing model training. We compare *NeRD* against its **Abs Pred** variant which predicts the absolute next state of the robot, in Fig. A.2(b). The results in the figure show that even for the low-dimensional system like *Double Pendulum*, predicting the absolute state significantly increases training difficulty and results in a prediction error $26\times$ larger than the error of predicting with relative state changes.

A.3.3.4 Robot-Centric State Representation

Next, we design experiments to demonstrate the importance of the robot-centric and spatially-invariant simulation state representation. For this ablation study, we train a model with the robot state and contact quantities represented in world space (**World Frame** variant in Fig. A.2(b)), *i.e.*, the loss formulation in Eq. 1. As shown in the results, using the world-frame representation does not degrade the model’s performance in the *Double Pendulum* case of contact-free motion. This is because the pendulum has a revolute base joint that remains fixed in position, limiting the visited states to the domain covered by the training dataset. In contrast, the world-frame representation fails entirely in the *Ant* running task, as the *Ant* moves far away from the origin during running and quickly reaches regions outside the training dataset’s distribution, causing the model to make unreliable predictions.

A.3.3.5 Model Input and Output Normalization

We then show the critical role of normalizing both the inputs and outputs of the neural robot dynamics models. As shown by the **No Inp. Norm** and **No Out. Norm** variants in Fig. A.2(b), removing either input or output normalization degrades the performance of the *NeRD* model. This is because the input normalization effectively regularizes the ranges of the inputs to the model, making model training stable and efficient. Meanwhile, output normalization mitigates the dominance of the high-magnitude and high-variance velocity terms in the loss function, balancing prediction accuracy across the different state dimensions.

A.3.3.6 History Window Size h

NeRD generally gains slight performance improvements when the history window size h increases. We provide a comparison on history window sizes $h = 1$, $h = 5$, and $h = 10$ in Fig. A.2(c). We

choose $h = 10$, as we find $h = 10$ consistently provides stable training and generally achieves the best performance across all tasks in our experiments. Though quite rare, we also notice that further increasing h (e.g., $h = 20$) will occasionally result in an exploded training loss.

References

- [1] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International conference on machine learning*, pages 4470–4479. PMLR, 2018.
- [2] M. Andriluka, B. Tabanpour, C. D. Freeman, and C. Sminchisescu. Learned neural physics simulation for articulated 3d human pose reconstruction. In *Computer Vision – ECCV 2024: 18th European Conference, Proceedings, Part LXXXIV*, page 320–336, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-72906-5.
- [3] K. R. Allen, T. L. Guevara, Y. Rubanova, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, and T. Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1157–1167. PMLR, 14–18 Dec 2023.
- [4] A. Karpathy. nanoGPT, 2023. URL <https://github.com/karpathy/nanoGPT>.
- [5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [6] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. Any-mal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 38–44, 2016.
- [7] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.
- [8] C. Li, A. Krause, and M. Hutter. Robotic world model: A neural network simulator for robust policy optimization in robotics, 2025.
- [9] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme. Neursim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, page 9474–9481. IEEE Press, 2021.
- [10] L. Fussell, K. Bergamin, and D. Holden. Supertrack: motion tracking for physically simulated characters using supervised learning. *ACM Trans. Graph.*, 40(6), Dec. 2021. ISSN 0730-0301.