

Continual Learning and Lifting of Koopman Dynamics for Linear Control of Legged Robots

Feihan Li

FEIHANL@ANDREW.CMU.EDU

Abulikemu Abuduweili

ABULIKEA@ANDREW.CMU.EDU

Yifan Sun

YIFANSU2@ANDREW.CMU.EDU

Rui Chen

RUIC3@ANDREW.CMU.EDU

Weiye Zhao

WEIYEZHA@ANDREW.CMU.EDU

Changliu Liu

CLIU6@ANDREW.CMU.EDU

Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract

The control of legged robots, particularly humanoid and quadruped robots, presents significant challenges due to their high-dimensional and nonlinear dynamics. While linear systems can be effectively controlled using methods like Model Predictive Control (MPC), the control of nonlinear systems remains complex. One promising solution is the Koopman Operator, which approximates nonlinear dynamics with a linear model, enabling the use of proven linear control techniques. However, achieving accurate linearization through data-driven methods is difficult due to issues like approximation error, domain shifts, and the limitations of fixed linear state-space representations. These challenges restrict the scalability of Koopman-based approaches. This paper addresses these challenges by proposing Incremental Koopman algorithm designed to iteratively refine Koopman dynamics for high-dimensional legged robots. The key idea is to progressively expand the dataset and latent space dimension, enabling the learned Koopman dynamics to converge towards accurate approximations of the true system dynamics. Theoretical analysis shows that the linear approximation error of our method converges monotonically. Experimental results demonstrate that our method achieves high control performance on robots like Unitree G1, H1, A1, Go2 and ANYmal D, across various terrains using simple linear MPC controllers. This work is the first to successfully apply linearized Koopman dynamics for locomotion control of high-dimensional legged robots, enabling a scalable model-based control solution.

Keywords: Koopman Operator Theory, Model Predictive Control, Legged Robots, Continual Learning

1. Introduction

The control of legged robots has garnered increasing attention due to the growing potential of humanoid and quadruped robots in real-world applications. However, the task is complicated with challenges due to the high-dimensional and nonlinear dynamics inherent in legged systems. Existing control methods for nonlinear systems exhibit a trade-off between controller synthesis and modeling accuracy, falling into model-free and model-based approaches. Model-free methods directly derive the control law without explicitly modeling the system dynamics. Reinforcement learning (RL) is the representative example, demonstrating notable success in controlling legged robots in real-world applications [Song et al. \(2021\)](#); [Haarnoja et al. \(2018\)](#); [Zhao et al.; Lee et al. \(2020\)](#); [He et al. \(2024\)](#); [Fu et al. \(2024\)](#); [Luo et al. \(2023\)](#). Despite their strong performance, model-free control laws are often highly task-specific and require retraining or fine-tuning when adapting to new tasks.

Model-based approaches decouple the modeling of system dynamics from the realization of task objectives, enabling more efficient adaptation to new tasks. The models used in these approaches can be categorized as: original nonlinear models, locally linearized models, and globally linearized models, with increasing modeling complexity and decreasing control complexity. For direct control of nonlinear models, [Yi et al. \(2024\)](#) introduced sampling-based MPC for legged robots, while [Kazemi et al. \(2013\)](#) applied robust backstepping control to quadruped robots. Local linearization methods, iLQR and NMPC often combine analytical [Sotaro Katayama and Tazaki \(2023\)](#); [Zhu et al. \(2024\)](#); [Zhang et al. \(2024\)](#) or learned models [Nagabandi et al. \(2017\)](#); [Liu et al. \(2023\)](#) with complex controllers. A promising alternative to these approaches is employing Koopman Operator Theory to construct globally linearized models of legged systems and enabling the use of a simpler controller, e.g., linear MPC, with performance guarantees. Although Koopman-based methods have proven effective in motion planning [Kim et al. \(2024a\)](#) and gait prediction [Krolicki et al. \(2022\)](#), no prior work has directly modeled the whole body dynamics of legged robots with Koopman Operator.

A Koopman Operator can be derived analytically or approximated through data-driven approaches. While analytical approaches are extremely challenging for high-dimensional nonlinear systems like legged robots, our method adopts a data-driven approach that reduces reliance on expert knowledge and enhances generalizability. Recent works have applied data-driven approaches to obtain linear models for high-level motion planning [Kim et al. \(2024a\)](#); [Lyu et al. \(2023\)](#). However, these methods typically depend on end-to-end optimization aimed at specific task objectives, which can limit their broader applicability. In contrast, [Shi and Meng \(2022\)](#); [Korda and Mezić \(2018b\)](#); [Mamakoukas et al. \(2021\)](#) have focused on obtaining accurate Koopman approximations for low-level control, but their applicability is limited to low-dimensional systems [Shi et al. \(2024\)](#). It remains challenging to apply Koopman Operator to linearize high-dimensional nonlinear systems due to (a) imperfect approximations and (b) domain shifts arising from limited state transition data that only covers a subspace of the entire state space. Such approximation errors make the system sensitive to inaccurately modeled dynamics, while domain shifts cause failures under perturbations.

To address these challenges, we propose an incremental Koopman algorithm tailored for high-dimensional legged robots, designed to continually learn and lift Koopman dynamics. The core strategy involves progressively expanding the training dataset and the dimension of the latent space. As the dataset and latent space grow, it naturally spans a state space where the refined Koopman dynamics achieve minimized linear approximation errors upon convergence. Additionally, we provide a theoretical analysis to validate the convergence of the learned Koopman dynamics towards the true Koopman dynamics. Experimental results demonstrate that our approach reduces linearization error after only a few iterations and enables effective locomotion control for legged robots using Model Predictive Control (MPC). Our key contributions are summarized as follows:

1. We propose an iterative continual learning algorithm to mitigate linear approximation errors for Koopman dynamics, supported by a theoretical guarantee of convergence.
2. Our approach is the first work to successfully implement locomotion control for high-dimensional legged robots using linearized system dynamics with linear MPC controllers.

2. Problem Formulation

2.1. Preliminary: Koopman Operator Theory

Koopman Operator Theory offers a powerful framework for analyzing nonlinear dynamical systems by mapping them into equivalent linear systems in an infinite-dimensional latent space. Consider

the discrete-time nonlinear autonomous system $s_{t+1} = f(s_t)$, where $s_t \in \mathcal{S}$ represents the state. Koopman operator theory introduces the embedding function $\phi : \mathcal{S} \rightarrow \mathcal{O}$ to map the original state space \mathcal{S} to an infinite dimensional latent space \mathcal{O} . In this latent space, the dynamics become linear through the Koopman Operator \mathcal{K} :

$$\mathcal{K}\phi(s) = \phi(f(s)) \quad (1)$$

Consider a non-autonomous dynamical system with control input $u_t \in \mathcal{U} \subset \mathbb{R}^{m'}$ and system state $x_t \in \mathcal{X} \subset \mathbb{R}^{n'}$, given by $x_{t+1} = f(x_t, u_t)$. The Koopman Operator transforms the system dynamics into a linear form.

$$\mathcal{K}\phi(x_t, u_t) = \phi(f(x_t, u_t)) = \phi(x_{t+1}) \quad (2)$$

In practice, we approximate the Koopman operator by constraining the latent space to a finite-dimensional vector space. For a non-autonomous system, a common strategy is to define the embedding function as $\phi(x_t, u_t) = [g(x_t); u_t]$, where $g : \mathbb{R}^{n'} \rightarrow \mathbb{R}^n$ is a state embedding function. The Koopman operator \mathcal{K} can be approximated by a matrix representation $K = \begin{bmatrix} A \in \mathbb{R}^{n \times n} & B \in \mathbb{R}^{n \times m'} \\ C \in \mathbb{R}^{m' \times n} & D \in \mathbb{R}^{m' \times m'} \end{bmatrix}$. In conjunction with (2), we obtain a linear model in the lifted space with the state evolution:

$$g(x_{t+1}) = Ag(x_t) + Bu_t \quad (3)$$

To preserve state information with clear semantics [Shi and Meng \(2022\)](#), and avoid the case of degeneration, such as cases where $A = B = 0$ and $g(x) \equiv 0$, we concatenate the original state with the neural network embedding to define the latent state z_t as $z_t = g(x_t) = [x_t, g'(x_t)]^\top$, where $g' : \mathbb{R}^{n'} \rightarrow \mathbb{R}^{n-n'}$ is a parameterized neural network. This approach allows us to retrieve the original state using a linear transformation.

$$x_{t+1} = Pz_{t+1} = Pg(x_{t+1}), \quad P = [I_{n' \times n'}, \mathbf{0}_{n' \times (n-n')}] \in \mathbb{R}^{n' \times n} \quad (4)$$

This formulation supports solving state-constrained control problems (e.g., collision avoidance) by directly transforming the constraints on x to constraints on z without compromising the lifted system's linear properties. Additionally, we use an end-to-end training approach for both the embedding function and the Koopman operator, parameterized as $\mathcal{T} \doteq (g, A, B)$.

2.2. Linear Model Predictive Control

With the linear model from the Koopman operator, in our high-dimensional legged robot control tasks, we employ a linear MPC algorithm as the controller π_{mpc} , thus the problem can be easily solved by Quadratic Programming (QP). The optimization formulation at time step t is as follows:

$$\begin{aligned} \min_{u_{t:t+H-1}} \quad & \|Pz_{t:t+H-1} - x_{t:t+H-1}^*\|_Q^2 + \|u_{t:t+H-1}\|_R^2 + \|Pz_{t+H} - x_{t+H}^*\|_F^2 \\ \text{s.t. } \quad & z_{t+k+1} = Az_{t+k} + Bu_{t+k}, \quad u_{t+k} \in [u_{min}, u_{max}], \quad \forall k = 0, \dots, H-1 \end{aligned} \quad (5)$$

where H denotes the horizon length, $x_{t:t+H}^*$ represents the reference trajectory, Q, R, F are cost matrices and $\|x\|_Q^2$ is defined as $x^T Q x$. The range $[u_{min}, u_{max}]$ specifies the valid interval for

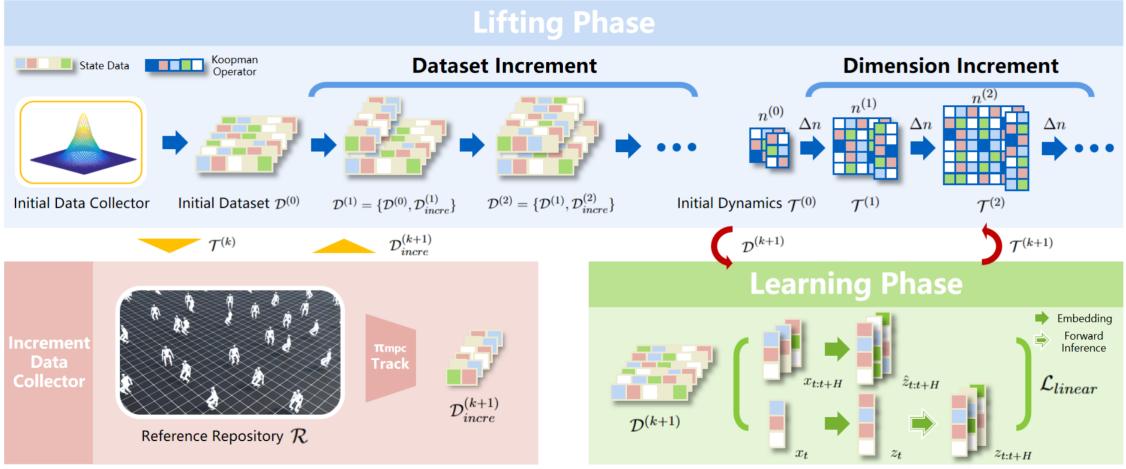


Figure 1: Overview of Incremental Koopman Algorithm

control inputs. The first constraint ensures adherence to the learned dynamics in the latent space, while the second constraint bounds control signals within the valid region. It should be noted that, unlike model-free methods, the decomposition of dynamics and controller makes full-body reference tracking essential for maintaining stable and reasonable movement patterns, as the MPC controller itself does not imply gait information.

2.3. Legged Robot System

As demonstrated in [Wieber et al. \(2016\)](#), legged robot systems inherently possess the following characteristics: (i) discontinuities resulting from mode transitions, (ii) sensitivity to noisy control inputs while maintaining balance, and (iii) failure situations that are often irrecoverable. Existing Koopman-based methods focus mainly on low-dimensional continuous systems with fixed contact modes [Shi and Meng \(2022\)](#), where the dynamics are continuous and easier to fit. However, when applied to high-dimensional hybrid legged systems, the complex and diverse contact modes introduce significant discontinuities, greatly complicating the modeling process. Moreover, these existing Koopman methods, which exhibit considerable approximation errors and limited latent subspace [Han et al. \(2023\)](#), render hybrid legged systems prone to irrecoverable failures caused by unmodeled dynamics and external perturbations. To overcome these challenges, we will introduce an algorithm that accurately and robustly models dynamics through continual learning and lifting, effectively expanding a resilient latent subspace.

3. Incremental Koopman Algorithm

As shown in Figure 1, the pipeline consists of two data collectors and two phases. After the initial data collection and dynamics learning, we iteratively alternate between the lifting phase, where the dataset is expanded by the increment data collector and the latent space dimension is increased, and the learning phase, during which dynamics are retrained based on the updated dataset and dimension.

3.1. Initialization and Increment of Dataset

Initial Data Collector. Unlike low-dimensional continuous systems explored in [Shi and Meng \(2022\)](#), the high-dimensional hybrid dynamics of legged robots need training data with reasonable

gait and consistent contact modes to form a meaningful latent subspace. Thus, we use an initial data collector (e.g., an RL policy or tele-operation) to generate dynamically feasible trajectories with reasonable movement patterns for the construction of initial dataset $\mathcal{D}^{(0)}$.

Increment Data Collector. Our method enables robust tracking within the reference repository \mathcal{R} , where references may be dynamically feasible or infeasible, with gaits and contact modes similar to $\mathcal{D}^{(0)}$. While $\mathcal{D}^{(0)}$ lays the foundation for the latent subspace, learning from limited samples—especially from a “good behavioral collector”—is insufficient. Models may still exhibit significant approximation errors in near-failure or failure scenarios, leading to ineffective tracking. Thus we use the MPC controller in Section 2.2 to track references in \mathcal{R} , collecting failed tracking data $\mathcal{D}_{\text{incre}}$ to naturally expand the dataset, thereby enhancing the robustness of the latent subspace.

Notably, our algorithm can be applied to any eligible initial data collector and reference repository \mathcal{R} . In this work, within a deterministic environment, we use a Proximal Policy Optimization (PPO [Schulman et al. \(2017b\)](#)) policy to build $\mathcal{D}^{(0)}$ and the initial reference repository $\mathcal{R}_{\text{init}}$, then add uniform noise in the range [-0.05, 0.05] to all references, generating the dynamically infeasible \mathcal{R} .

3.2. Training with Given Data

Based on the base Koopman framework introduced in Section 2.1, our primary goal is to learn the embedding function g and the Koopman operators A, B end-to-end. To achieve this, we employ a discounted k-step prediction loss to enhance long-horizon prediction capabilities which is crucial for MPC control. Given a set of trajectory data $\{x_{t:t+H}; u_{t:t+H-1}\}$, real latent trajectories can be computed as $\{z_{t:t+H} = g(x_{t:t+H})\}$. At the same time, predicted latent trajectories can be obtained through inference: $\{\hat{z}_{t:t+H} | \hat{z}_t = z_t, \hat{z}_{t+h} = A\hat{z}_{t+h-1} + B u_{t+h-1}, h = 1, 2, \dots, H\}$. Thus the loss function is defined as:

$$\mathcal{L}_{\text{koopman}} = \frac{1}{H} \sum_{h=1}^H \gamma^h \left(\underbrace{\|\hat{z}_{t+h} - z_{t+h}\|^2}_{\mathcal{L}_{\text{linear}}} + \alpha \underbrace{\|\hat{x}_{t+h} - x_{t+h}\|^2}_{\mathcal{L}_{\text{recon}}} \right) \quad (6)$$

where $\hat{x}_{t:t+H} = P\hat{z}_{t:t+H}$, α is the weight of $\mathcal{L}_{\text{recon}}$ and γ is the discount factor. Here $\mathcal{L}_{\text{linear}}$ focuses on the linearization effect, while $\mathcal{L}_{\text{recon}}$ addresses on the reconstruction effect. Thus can derive dynamics $\mathcal{T}^{(k)} \doteq (g_{n^{(k)}}^{(k)}, A_{n^{(k)}}^{(k)}, B_{n^{(k)}}^{(k)})$ for dataset $\mathcal{D}^{(k)}$, here $n^{(k)}$ represents the latent space dimension of iteration k .

3.3. Continual Learning and Lifting

The incremental Koopman algorithm begins with the initial dynamics $\mathcal{T}^{(0)}$ trained on the initial dataset $\mathcal{D}^{(0)}$. During the lifting phase, we use the increment data collector introduced in Section 3.1 to gather the incremental dataset $\mathcal{D}_{\text{incre}}^{(1)}$, which is then used to augment the initial dataset, forming $\mathcal{D}^{(1)} = \mathcal{D}^{(0)} \cup \mathcal{D}_{\text{incre}}^{(1)}$. Simultaneously, to accommodate the growing dataset, the latent space dimension is updated as $n^{(1)} = n^{(0)} + \Delta n$, where Δn is a hyperparameter chosen based on the trade-off between computational complexity, performance gain, and the requirement $m = \Omega(n \log(n))$ to be introduced in Section 3.4, where m denotes the dataset size.

In the learning phase, we utilize the loss introduced in Section 3.2 to obtain $\mathcal{T}^{(1)} = (g_{n^{(1)}}^{(1)}, A_{n^{(1)}}^{(1)}, B_{n^{(1)}}^{(1)})$. This process is repeated until the survival steps T_{sur} under MPC control with dynamics $\mathcal{T}^{(k)}$ cease to show improvement over $\mathcal{T}^{(k-1)}$. Check Algorithm 1 for details. While sampling all

state-action pairs is unrealistic and unscalable in high-dimensional problems, our method balances sample complexity, approximation quality, and scalability. It leverages on-policy exploration to gradually minimize approximation errors within the MPC controller’s region of attraction, enabling reliable decisions and establishing robust linear latent dynamics. We validate our claims in Section 4.

3.4. Theoretical analysis

Incremental increases in the dimension of latent space and data size are central to our proposed algorithm. This section demonstrates that, under our incremental strategy, the learned Koopman operator matrix K converges to the true Koopman operator \mathcal{K} . For theoretical analysis, we primarily focus on autonomous systems as described in eq. (1). For control systems, the methodology extends by setting $s = [x; u]^\top$. The fundamental premise, as established in Korda and Mezić (2018a,b), is that the learned Koopman operator K is the L_2 projection of the true Koopman operator \mathcal{K} onto the span of the n -dimensional embedding functions $\phi(s) = [\phi_1(s), \dots, \phi_n(s)]$, where $\phi_i(s)$ is the one-dimensional embedding function for the i -th dimension of the latent space. Unlike the convergence analysis in Korda and Mezić (2018a), our theoretical framework introduces the convergence rate of our incremental Koopman algorithm. For a detailed proof, please refer to appendix D.

Theorem 1. *Under the assumptions of 1) data samples s_1, \dots, s_m are i.i.d distributed; 2) the latent state is bounded, $\|\phi(s)\| < \infty$; 3) the embedding functions ϕ_1, \dots, ϕ_n are orthogonal (independent).*

(a) *The learned Koopman operator converges to the true Koopman Operator:*

$$\lim_{m=\Omega(n \ln(n)), n \rightarrow \infty} K \rightarrow \mathcal{K}. \quad (7)$$

(b) *Assuming additional conditions: 4) The eigenvalues of \mathcal{K} decay sufficiently fast, such as $|\lambda_i| \leq \frac{C}{i^\alpha}$ for some constant $C > 0, \alpha > \frac{1}{2}$. 5) The embedding functions $\phi(\cdot) = [\phi_1, \dots, \phi_n]^\top$ correspond to the first n eigenfunctions of \mathcal{K} associated with the largest eigenvalues in magnitude. Then the convergence rate of the linear approximation error is given by:*

$$\text{error} \leq \mathcal{O}\left(\sqrt{\frac{\ln(n)}{m}}\right) + \mathcal{O}\left(\frac{1}{n^{\alpha-\frac{1}{2}}}\right). \quad (8)$$

As indicated in Theorem 1, the proposed algorithm, by incrementally increasing the latent dimension n and data size m , progressively reduces the linear approximation error. According to Theorem 1(b), if the sample size meets $m = \Omega(n \ln(n))$, this reduction error follows the rate of $\mathcal{O}(n^{-\frac{1}{2}} + n^{\frac{1}{2}-\alpha})$ as the dimension increases. Notably, for $\alpha = 1$, the convergence rate becomes to $\mathcal{O}(n^{-\frac{1}{2}})$. In our experiments, we adhere to this guideline to ensure adequate sample generation.

4. Experiments

In our experiments, we evaluate the proposed method by addressing the following questions: **Q1** Does the learned Koopman Dynamics from algorithm 1 achieve better state prediction performance compared to baselines? **Q2** Does our approach, combining learned Koopman dynamics with MPC control, achieve superior tracking performance relative to baselines? **Q3** Is the continual expansion of the dataset size in our proposed method effective? **Q4** Is the continual increase in the dimensions of the latent state beneficial?

4.1. Experiment Setup

Task settings. Our experiments use IsaacLab [Mittal et al. \(2023\)](#) as the simulation platform, the most advanced simulator offering comprehensive and flexible support for a wide range of robots and environments. Five different legged robots are included in our experiments: (i) **ANYmal-D:** (Figure 8(a)) A quadruped robot ($\mathcal{U} \subseteq \mathbb{R}^{12}$) designed by ANYbotics. (ii) **Unitree-A1:** (Figure 8(b)) A quadruped robot ($\mathcal{U} \subseteq \mathbb{R}^{12}$) designed by Unitree. (iii) **Unitree-Go2:** (Figure 8(b)) A quadruped robot ($\mathcal{U} \subseteq \mathbb{R}^{12}$) designed by Unitree. (iv) **Unitree-H1:** (Figure 8(b)) A Humanoid ($\mathcal{U} \subseteq \mathbb{R}^{19}$) designed by Unitree. (v) **Unitree-G1:** (Figure 8(b)) A Humanoid ($\mathcal{U} \subseteq \mathbb{R}^{23}$) designed by Unitree. Furthermore, two different types of terrain are considered, including (i) **Flat:** (Figure 8(f)) flat terrain. (ii) **Rough:** (Figure 8(g)) rough terrain with random surface irregularities, where the height follows a uniform distribution between 0.005 and 0.025, and the minimum height variation is 0.005 (in m). All test suites are based on the task **Walk:** tracking a velocity command with uniformly sampled heading direction, x-axis linear velocity, and y-axis linear velocity. Considering these settings, we design 7 test suites with 5 types of legged robots and 2 types of terrain, which are summarized in Table 3. We name these test suites as {Terrain}-{Make}-{Model}.

Comparison Group. We compare Incremental Koopman algorithm with state-of-the-art Koopman-based algorithms (i) Deep KoopmanU with Control (DKUC, [Shi and Meng \(2022\)](#)) algorithm, (ii) Deep Koopman Affine with control (DKAC, [Shi and Meng \(2022\)](#)) algorithm, (iii) Neural Network Dynamics Model (NNDM, [Nagabandi et al. \(2017\)](#); [Liu et al. \(2023\)](#)) with MPC (iv) Deep Koopman Reinforcement Learning (DKRL, [Song et al. \(2021\)](#)). It worth noting that all methods are trained separately on 7 test suites. And for all experiments, we take the best algorithm-specific parameters mentioned in the original paper and keep the common parameters as the same.

Metrics. We introduce the following metrics to evaluate the tracking ability of our algorithm: (i) **Prediction-based:** we leverage k-step prediction error $E_{pre}(k) \doteq \frac{1}{kn'} \sum_{t=1}^k \|x_t - x_t^*\|_1$ ($x_0 = x_0^*$) generally used for measuring prediction ability of given dynamics. (ii) **Pose-based:** To evaluate tracking accuracy, we evaluate Joint-relative mean per-joint position error (E_{JrPE}), Joint-relative mean per-joint velocity error (E_{JrVE}), Joint-relative mean per-joint acceleration error (E_{JrAE}), Root mean position error (E_{RPE}), Root mean orientation error (E_{ROE}), Root mean linear velocity error (E_{RLVE}) and Root mean angular velocity error (E_{RLAE}). Check Appendix E.4 for detailed equations. (iii) **Physics-based:** Considering the easy-to-fail feature of legged robots, for the same reference repository \mathcal{R} , we evaluate the average survival simulation time step T_{Sur} (set 200 as upper bound, simulation frequency is 50Hz) for each algorithms, deeming tracking unsuccessful when E_{JrPE} is larger than ϵ_{fail} . Check Table 3 for details.

State. The system state $x_t \doteq \begin{cases} (j_t, \dot{j}_t, p_t^z, \dot{p}_t, \dot{r}_t), & \text{for humanoid} \\ (j_t, \dot{j}_t, p_t^z, \dot{p}_t, r_t, \dot{r}_t), & \text{for quadruped robot} \end{cases}$ of dynamics and controller includes the joint position j_t and joint velocity \dot{j}_t in local coordinates, root height p_t^z , 3D root linear velocity p_t , 4D root orientation r_t (optional for humanoid and represented as a quaternion), and 3D root angular velocity \dot{r}_t in world coordinates. All of the aforementioned quantities are normalized to follow a standard Gaussian distribution, $\mathcal{N}(0, 1)$, in order to eliminate inconsistencies in the scale of different physical quantities.

Control Input and Low-level Controller. We use a proportional-derivative (PD) controller at each joint of the legged robots as the low-level controller. Thus, the control input u_t determined by the MPC serves as the joint target, denoted as $j_t^d = u_t$. The torque applied to each joint is given by

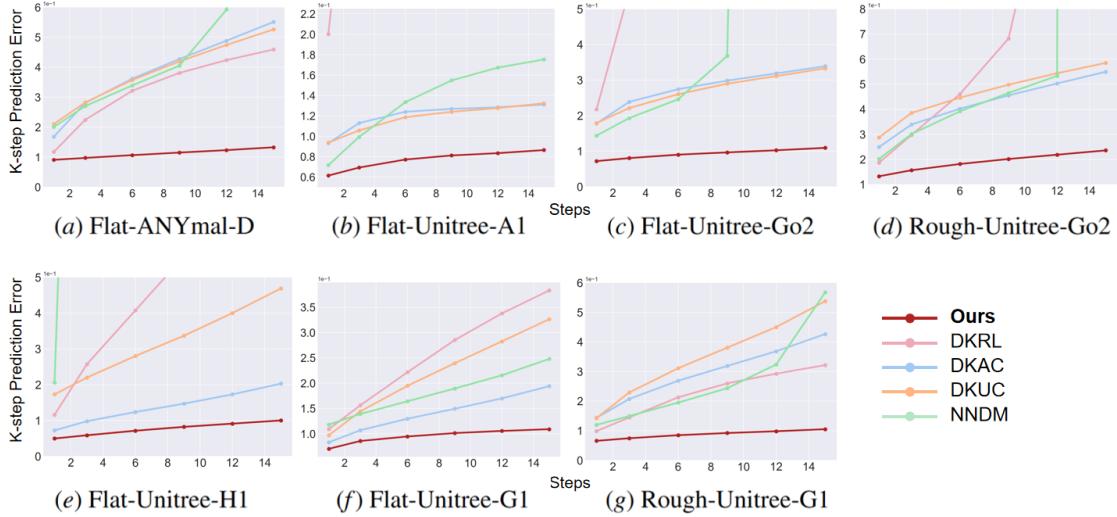


Figure 2: Comparison results of k-step prediction error in our test suites.

$\tau_t = k^p \circ (j_t^d - j_t) - k^d \circ \dot{j}_t$, where k^p and k^d are the proportional and derivative gains, respectively. The low-level controller operates at a frequency of 200 Hz, with a decimation factor of 4.

4.2. K-step Prediction Error

To demonstrate the superiority of our algorithms, we evaluate the k-step prediction performance across the seven test suites mentioned above. Specifically, we set k to $[1, 3, 6, 9, 12, 15]$ and compute $E_{pre}(k)$ for each algorithm, subsequently plotting the results in Figure 2. The results highlight our algorithm’s exceptional ability to maintain low prediction errors, even over long horizons in complex, nonlinear tasks—challenging for other algorithms in our comparison. Figures 3(c) to 3(e) show that NNDM and DKRL struggle with the accumulation of explosive errors as horizon length k increases, while our algorithm remains stable, with negligible error growth. Though DKAC and DKUC manage a stable rate of loss increase, they fail to infer boundary cases of complex robot motions due to limitations in subspace modeling. This is further confirmed by their poor performance in noisy, complex tracking experiments, as discussed in Section 4.3. These results address Q1.

4.3. Tracking Performance



Figure 3: Visualization of tracking performance on Flat-Unitree-G1

To highlight our lead in tracking tasks, we track 3e3 references in \mathcal{R} over 200 steps in parallel, presenting the results in Table 1 with average tracking metrics from Section 4.1. Further details are provided in Appendix E.5.

Algorithm	Synthesised Tracking Metrics							
	Joint-relative ↓			Root-relative ↓			Survival ↑	
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0348	0.6499	43.1465	0.1231	0.0668	0.1216	0.3289	188.4514
DKRL	0.0823	1.1251	68.9520	0.2978	0.1561	0.2089	0.5634	116.9540
DKAC	0.1816	2.0694	117.5515	0.3955	0.2749	0.2888	0.9143	25.0254
DKUC	0.1576	1.0828	50.5745	0.2934	0.1989	0.2252	0.5559	82.4620
NNDM	0.1439	2.2020	127.4524	0.4334	0.2506	0.2996	0.8536	35.4709

Table 1: The average tracking metrics evaluated for each algorithm across all 7 test suites.

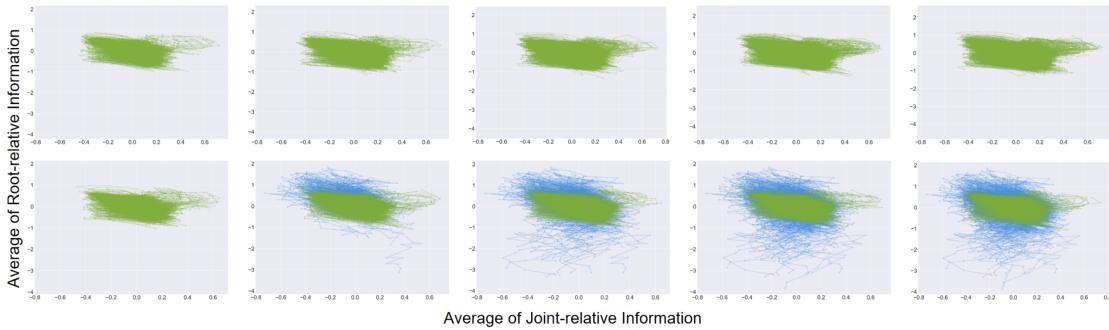


Figure 4: Visualization of the data distribution by plotting the means of the joint-relative and root-relative states. The first row illustrates the dataset expansion achieved using an RL policy, while the second row shows the results from applying our algorithm for dataset expansion.

Compared to other algorithms, our approach shows (i) high T_{Sur} values nearing the upper bound of 200, (ii) precise, continuous joint-level tracking, and (iii) sustained global tracking with balance and stability. All measured metrics are significantly lower for competing algorithms, which often fail to track correctly and fall after only a few attempts. In contrast, our algorithm delivers efficient, accurate tracking with a steady gait, demonstrating its ability to infer corner cases during tracking, thanks to the continual learning and lifting process in Algorithm 1. Meanwhile, others struggle to achieve the correct poses and gait when faced with falling. Notably, the 200-step upper bound for T_{Sur} suggests that many test cases remain robust enough for even longer tracking durations.

Although DKRL and DKUC achieve relatively high T_{Sur} , their E_{JrPE} and E_{RPE} are 2 to 5 times higher than ours, indicating that their metrics include brief failure episodes. In contrast, our algorithm maintains robust tracking throughout, showcasing superior steady tracking with minimal error. DKAC and NNDM, however, suffer from short inference capabilities and sensitivity to the dynamics’ corner cases, making them prone to failure—especially in situations where recovery is impossible for legged robots. As shown in Figure 3, unrecoverable failures end the tracking process, answering Q2.

4.4. Ablation on Dataset Increment

For the ablation study on dataset increment, we select Flat-Unitree-Go2 and Flat-Unitree-G1 to examine the impact of dataset increment techniques. As shown in Table 2 and fig. 5, the k-step prediction error rises sharply with increasing steps, highlighting limited inferencing capability when

Algorithm	Average Tracking Metrics of Flat-Unitree-Go2 and Flat-Unitree-G1							
	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Original	0.0246	0.5954	42.1403	0.0673	0.0245	0.0650	0.2060	196.6190
w/o Data.I.	0.2061	1.5251	65.5157	0.3452	0.2449	0.2740	0.6303	53.0540
w/o Dim.I.	0.1189	1.1743	63.6319	0.2526	0.1936	0.2178	0.5782	100.9350

Table 2: The average tracking metrics evaluated for the ablation studies. ‘w/o Data.I.’ and ‘w/o Dim.I.’ indicate the performance of the original method without data increment and dimension increment techniques, respectively

dynamics encounter corner cases due to insufficient latent subspace modeling. Furthermore, E_{JrPE} is nearly seven times higher than the original method, indicating frequent failure scenarios. The low survival steps T_{Sur} further corroborate this observation. In contrast, the original algorithm Algorithm 1 maintains exceptional tracking performance and loss control, demonstrating the effectiveness of dataset enlargement. Additionally, Figure 4 visualizes the data enlargement process, showing that RL policy-based expansion is ineffective due to repetitive data, while our algorithm achieves effective dataset growth. This underscores the importance of robust subspace modeling and accurate inferencing, addressing Q3.

4.5. Ablation on Dimension Increment

To conduct the ablation study on dimension increment, we select the same test suites mentioned in Section 4.4. As shown in Figure 5, the k-step prediction error remains steady and relatively low, similar to the original methods, benefiting from the continual increment of the dataset to model the subspace. However, as indicated in Table 2, although the algorithm without dimension increment outperforms the algorithm without data increment, its tracking performance still falls significantly short of the original method. This suggests that a low latent space dimension fails to model robust dynamics strong enough to resist unseen noise during simulation. Furthermore, the lack of inferencing capability highlights its shortcomings in linearization and generalization. Thus, the dimension increment technique emphasizes the importance of modeling robust and general sub-dynamics, addressing Q4.

5. Conclusion and Future Works

By progressively expanding the dataset and latent subspace, our method ensures convergence of linearization errors and enables accurate approximations of the system dynamics. Experimental results show that the approach achieves high control performance with simple MPC controllers on various tasks after just a few iterations. This work represents the first successful application of linearized Koopman dynamics to locomotion control of legged robots, offering a scalable and model-based control solution. In the future, tele-operation or retargeted human data will be tested to implement our algorithm in real-world scenarios.

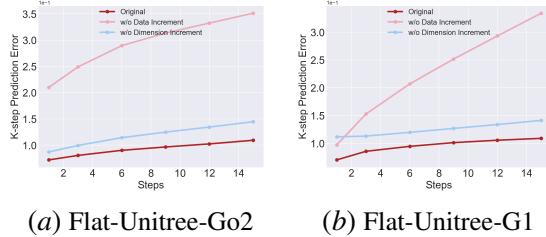


Figure 5: K-step prediction error comparison of ablation on dimension and dataset increment.

References

- Abulikemu Abuduweili and Changliu Liu. Robust online model adaptation by extended kalman filter with exponential moving average and dynamic multi-epoch strategy. In *Learning for Dynamics and Control*, pages 65–74. PMLR, 2020.
- Abulikemu Abuduweili and Changliu Liu. Online model adaptation with feedforward compensation. In *Conference on Robot Learning*, pages 3687–3709. PMLR, 2023.
- Hongyi Chen, Abulikemu Abuduweili, Aviral Agrawal, Yunhai Han, Harish Ravichandar, Changliu Liu, and Jeffrey Ichnowski. Korol: Learning visualizable object feature with koopman operator rollout for manipulation. *arXiv preprint arXiv:2407.00548*, 2024.
- Thor I Fossen and Aslaug Grovlen. Nonlinear output feedback control of dynamically positioned ships using vectorial observer backstepping. *IEEE transactions on control systems technology*, 6(1):121–128, 1998.
- Zipeng Fu, Qingqing Zhao, Qi Wu, Gordon Wetzstein, and Chelsea Finn. Humanplus: Humanoid shadowing and imitation from humans, 2024. URL <https://arxiv.org/abs/2406.10454>.
- Lars Grüne, Jürgen Pannek, Lars Grüne, and Jürgen Pannek. *Nonlinear model predictive control*. Springer, 2017.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.
- Yunhai Han, Mandy Xie, Ye Zhao, and Harish Ravichandar. On the utility of koopman operator theory in learning dexterous manipulation skills. In *Conference on Robot Learning*, pages 106–126. PMLR, 2023.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL <https://arxiv.org/abs/1512.03385>.
- Tairan He, Zhengyi Luo, Wenli Xiao, Chong Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Learning human-to-humanoid real-time whole-body teleoperation. *arXiv preprint arXiv:2403.04436*, 2024.
- Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven approximations of dynamical systems operators for control. *The Koopman operator in systems and control: concepts, methodologies, and applications*, pages 197–234, 2020.
- Hamed Kazemi, Vahid Johari Majd, and Majid M. Moghaddam. Modeling and robust backstepping control of an underactuated quadruped robot in bounding motion. *Robotica*, 31(3):423–439, 2013. doi: 10.1017/S0263574712000458.
- Jeonghwan Kim, Yunhai Han, Harish Ravichandar, and Sehoon Ha. Learning koopman dynamics for safe legged locomotion with reinforcement learning-based controller. *arXiv preprint arXiv:2409.14736*, 2024a.

- Jin Sung Kim, Ying Shuai Quan, and Chung Choo Chung. Uncertainty quantification of autoencoder-based koopman operator. In *2024 American Control Conference (ACC)*, pages 4631–4636. IEEE, 2024b.
- Milan Korda and Igor Mezić. On convergence of extended dynamic mode decomposition to the koopman operator. *Journal of Nonlinear Science*, 28:687–710, 2018a.
- Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018b.
- Alexander Krolicki, Dakota Rufino, Andrew Zheng, Sriram S.K.S Narayanan, Jackson Erb, and Umesh Vaidya. Modeling quadruped leg dynamics on deformable terrains using data-driven koopman operators. *IFAC-PapersOnLine*, 55(37):420–425, 2022. ISSN 2405-8963. doi: <https://doi.org/10.1016/j.ifacol.2022.11.219>. URL <https://www.sciencedirect.com/science/article/pii/S2405896322028622>. 2nd Modeling, Estimation and Control Conference MECC 2022.
- Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.
- Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 222–229. SciTePress, 2004.
- Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.
- Ziang Liu, Genggeng Zhou, Jeff He, Tobia Marcucci, Li Fei-Fei, Jiajun Wu, and Yunzhu Li. Model-based control with sparse neural dynamics, 2023. URL <https://arxiv.org/abs/2312.12791>.
- Zhengyi Luo, Jinkun Cao, Kris Kitani, Weipeng Xu, et al. Perpetual humanoid control for real-time simulated avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10895–10904, 2023.
- Xubo Lyu, Hanyang Hu, Seth Siriya, Ye Pu, and Mo Chen. Task-oriented koopman-based control with contrastive encoder. In *Conference on Robot Learning*, pages 93–105. PMLR, 2023.
- Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.
- Giorgos Mamakoukas, Maria L Castano, Xiaobo Tan, and Todd D Murphrey. Derivative-based koopman operators for real-time control of robotic systems. *IEEE Transactions on Robotics*, 37(6):2173–2192, 2021.
- Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot

learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017. URL <https://arxiv.org/abs/1708.02596>.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=HyxAfnA5tm>.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017b. URL <https://arxiv.org/abs/1707.06347>.

Haojie Shi and Max Q-H Meng. Deep koopman operator with control for nonlinear systems. *IEEE Robotics and Automation Letters*, 7(3):7700–7707, 2022.

Lu Shi, Masih Haseli, Giorgos Mamakoukas, Daniel Bruder, Ian Abraham, Todd Murphrey, Jorge Cortes, and Konstantinos Karydis. Koopman operators in robot learning, 2024. URL <https://arxiv.org/abs/2408.04200>.

Lixing Song, Junheng Wang, and Junhong Xu. A data-efficient reinforcement learning method based on local koopman operators. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 515–520, 2021. doi: 10.1109/ICMLA52953.2021.00086.

Masaki Murooka Sotaro Katayama and Yuichi Tazaki. Model predictive control of legged and humanoid robots: models and algorithms. *Advanced Robotics*, 37(5):298–315, 2023. doi: 10.1080/01691864.2023.2168134. URL <https://doi.org/10.1080/01691864.2023.2168134>.

Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12:389–434, 2012.

Pierre-Brice Wieber, Russ Tedrake, and Scott Kuindersma. Modeling and control of legged robots. In *Springer Handbook of Robotics, 2nd Ed.*, 2016. URL <https://api.semanticscholar.org/CorpusID:6790710>.

Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, page 1307–1346, Dec 2015. doi: 10.1007/s00332-015-9258-5. URL <http://dx.doi.org/10.1007/s00332-015-9258-5>.

Zeji Yi, Chaoyi Pan, Guanqi He, Guannan Qu, and Guanya Shi. Covo-mpc: Theoretical analysis of sampling-based mpc and optimal covariance design, 2024. URL <https://arxiv.org/abs/2401.07369>.

Peng Yin, Abulikemu Abuduweili, Shiqi Zhao, Lingyun Xu, Changliu Liu, and Sebastian Scherer. Bioslam: A bioinspired lifelong memory system for general place recognition. *IEEE Transactions on Robotics*, 2023.

Zixin Zhang, John Z. Zhang, Shuo Yang, and Zachary Manchester. Robots with attitude: Singularity-free quaternion-based model-predictive control for agile legged robots, 2024. URL <https://arxiv.org/abs/2409.09940>.

Weiye Zhao, Feihan Li, Yifan Sun, Rui Chen, Tianhao Wei, and Changliu Liu. Absolute policy optimization: Enhancing lower probability bound of performance with high confidence. In *Forty-first International Conference on Machine Learning*.

James Zhu, J. Joe Payne, and Aaron M. Johnson. Convergent ilqr for safe trajectory planning and control of legged robots, 2024. URL <https://arxiv.org/abs/2304.00346>.

Appendix A. Related Works

Controlling Nonlinear Dynamics. Nonlinear dynamic systems are generally controlled using two main approaches: model-free and model-based methods. Model-free methods, often associated with learning-based approaches, have become popular for addressing complex, high-dimensional systems in a data-driven manner. Reinforcement learning (RL) techniques, for instance, approximate a combination of system dynamics and control policies by training neural networks using methods like policy gradients or value iteration. Off-policy algorithms such as SAC [Haarnoja et al. \(2018\)](#) efficiently reuse data to learn control policies, whereas on-policy methods like PPO [Schulman et al. \(2017a\)](#) and its extension APO [Zhao et al.](#) ensure monotonic policy improvement. Another learning paradigm, imitation learning [Luo et al. \(2023\); He et al. \(2024\)](#), leverages large expert datasets to guide policy development. Despite their successes, these methods often suffer from challenges such as sample inefficiency, high computational cost, and limited generalization.

Model-based methods, on the other hand, rely on leveraging system physics to construct accurate models. Techniques like iLQR [Li and Todorov \(2004\)](#) and NMPC [Grüne et al. \(2017\); Liu et al. \(2023\)](#) use these models to optimize control trajectories, while methods such as backstepping [Fossen and Grovlen \(1998\)](#) are well-suited for systems with well-defined dynamics. Neural network-based approaches, such as [Nagabandi et al. \(2017\)](#), extend these ideas by approximating nonlinear dynamics for control tasks. Moreover, model-based RL combines data-driven techniques with physical modeling to describe nonlinear system dynamics. For instance, [Song et al. \(2021\)](#) uses RL workflows to fit multiple local approximations of global dynamics, while [Nagabandi et al. \(2017\)](#) employs RL reward functions to learn dynamics from extensive random-shooting data. However, even these approaches face significant challenges when applied to high-dimensional systems, such as legged robots, where designing accurate models becomes increasingly complex and task-specific.

Koopman Operator Theory. While model-based methods struggle with complex, high-dimensional systems and model-free methods lack interpretability and flexibility, linearizing nonlinear dynamics has recently emerged as a promising approach. Among these, Koopman Operator Theory has garnered attention for its ability to represent nonlinear dynamics in a higher-dimensional linear space. Current research on Koopman theory in control can be categorized into three main directions:

(i) Koopman Operator as a Feature Encoder: The Koopman operator has been applied in some methods to predict future states based on the current state [Han et al. \(2023\)](#) or vision inputs [Chen et al. \(2024\)](#). However, these approaches primarily serve as feature extractors and do not fully exploit the benefits of linearization. Additionally, designing an effective Koopman operator is often challenging due to the requirement for infinite-dimensional representations and the inefficiency of available data [Kaiser et al. \(2020\)](#).

(ii) Integrating Koopman Operators with RL: Other methods combine the Koopman latent dynamic space with traditional RL policies, either hierarchically [Kim et al. \(2024a\)](#) or end-to-end [Lyu et al. \(2023\)](#). While this integration aims to benefit from Koopman's properties, it still inherits the challenges of learning-based methods, particularly in terms of interpretability and generalization.

(iii) Linear Control in Koopman Latent Space: A promising direction applies traditional linear control methods, such as LQR or MPC, to the Koopman-linearized model, emphasizing interpretability and flexibility. For example, [Shi and Meng \(2022\)](#) applies LQR to a deep-learned Koopman operator, while others use radial basis functions (RBF) [Korda and Mezić \(2018b\)](#) or derivative basis functions [Mamakoukas et al. \(2021\)](#) to design the operator. Despite these advances, current methods

struggle to control high-dimensional systems, like legged robots due to severe model mismatch and domain shifts.

Finally, research has begun addressing the errors that arise during the Koopman dynamics learning process. Works such as [Kim et al. \(2024b\)](#) and [Korda and Mezić \(2018a\)](#) evaluate the reconstruction and projection errors, providing valuable theoretical foundations for further improvement.

Continual Learning. Continual learning aims to incrementally update knowledge from streaming data, often utilized in learning prediction or dynamics models as seen in works like [Nagabandi et al. \(2019\)](#); [Abuduweili and Liu \(2020\)](#). A prevalent approach in continual learning is memory replay [Rolnick et al. \(2019\)](#), where selected samples are stored and repeatedly replayed to train the model through methods such as random sampling [Rolnick et al. \(2019\)](#) or importance sampling [Yin et al. \(2023\)](#); [Abuduweili and Liu \(2023\)](#). From the memory replay perspective, our approach adopts a random replay strategy to efficiently generate and reuse data. Another relevant approach is the use of dynamic architecture-based methods, exemplified by Learning without Forgetting (LwF) [Li and Hoiem \(2017\)](#) and PackNet [Mallya and Lazebnik \(2018\)](#), which modify the neural network architecture incrementally during training. In contrast to these methods, which typically add parameters in the final layers for task-specific learning, our method leverages a theory-guided strategy to expand the latent state dimension.

Appendix B. Algorithm

Algorithm 1 presents the proposed the Incremental Learning algorithm for Koopman Operators.

Appendix C. Additional Experiments

C.1. Computational Cost Comparison

We compare the resource usage of all algorithms in terms of GPU and CPU consumption, as well as wall-clock time, using DKUC as a benchmark, as presented in Figure 6. The horizontal axis in the upper area represents the percentage of resource usage relative to DKUC, while the horizontal axis in the lower area displays the running time (in seconds). The experiments are based on the averages from all seven test suites set for tracking experiments, as described in Section 4.3, and were run and averaged over three different seeds. Our algorithms utilize nearly the same GPU and CPU resources as the baseline methods while demonstrating a sharp lead in k-step prediction and tracking experiments. Furthermore, although wall-clock time increases when compared to DKUC and DKAC methods, it remains low enough to facilitate real-time inference for efficient control of real robots while NNDM method spends unacceptable reasoning time, demonstrating the effectiveness of the Koopman method and our algorithms, thereby addressing Q5.

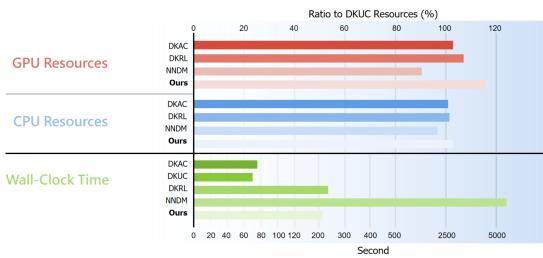


Figure 6: Computational cost(GPU occupancy, CPU occupancy and wall-clock time) comparison

Algorithm 1 Incremental Koopman Algorithm

Input: MPC Controller π_{mpc} , Initial latent space dimension $n^{(0)}$, Initial dynamics $\mathcal{T}^{(0)} = (g_n^{(0)}, A_n^{(0)}, B_n^{(0)})$, Initial dataset $\mathcal{D}^{(0)}$, Reference repository \mathcal{R} , Dimension increment step size Δn and Initial training epochs $J^{(0)}$

function: TrainKoopman(n, \mathcal{D}, J):

- optimizaer* \leftarrow Adam();
- scheduler* \leftarrow CosineAnnealingLR();
- $\mathcal{T} \doteq (g_n, A_n, B_n)$; {Initialization of Koopman dynamics}
- for** $k = 0, 1, 2, \dots, J - 1$ **do**
- for** \mathcal{B} in \mathcal{D} **do**
- $\mathcal{B}_z \leftarrow g(\mathcal{B})$; {Embed $x \in \mathcal{B}$ into latent vectors z }
- $l_k \leftarrow \mathcal{L}_{koopman}(\mathcal{T}, \mathcal{B}, \mathcal{B}_z)$; {Refer to Equation (6) for $\mathcal{L}_{koopman}$ }
- optimizer.zero_grad()*;
- $l_k.backward()$;
- optimizer.step()*;
- end for**
- scheduler.step()*;
- end for**
- return* \mathcal{T} ;

end function

for $j = 0, 1, 2, \dots$ **do**

- $n^{(j+1)} \leftarrow n^{(j)} + \Delta n$; {Increase latent space dimension}
- $\mathcal{D}_{incre}^{(j+1)} \leftarrow \pi_{mpc}(\mathcal{T}^{(j)}, \mathcal{R})$;
- $\mathcal{D}^{(j+1)} \leftarrow \{\mathcal{D}^{(j)}, \mathcal{D}_{incre}^{(j+1)}\}$; {Dataset augmentation}
- $J^{(j+1)} \leftarrow J^{(j)}$;
- $\mathcal{T}^{(j+1)} \leftarrow \text{TrainKoopman}(n^{(j+1)}, \mathcal{D}^{(j+1)}, J^{(j+1)})$;
- while** TrainKoopman failed **do**
- $J^{(j+1)} \leftarrow \frac{J^{(j+1)}}{2}$; {Adjust epochs when collapsing}
- $\mathcal{T}^{(j+1)} \leftarrow \text{TrainKoopman}(n^{(j+1)}, \mathcal{D}^{(j+1)}, J^{(j+1)})$;

end while

if TrackConverge($\mathcal{T}^{(j+1)}, \mathcal{T}^{(j)}$) **then**

- Break*; {Quit if tracking performance converge}

end if

end for

C.2. Evaluation of the theorem

We conducted experiments on the G1 robot to evaluate the conclusions of Theorem 1 and Theorem A.3. Figure Figure 7(a) illustrates the prediction error versus sample size with a fixed embedding dimension of $n = 500$. Consistently, the prediction error decreases as the sample size increases, aligning with the decreasing rate outlined in Equation (33). Figure Figure 7(b) displays the prediction error versus embedding dimension (or Koopman dimension) for a fixed sample size of $m = 180,000$. Similarly, an increase in the embedding dimension results in a decrease in prediction error, echoing the trend described in Equation (49).

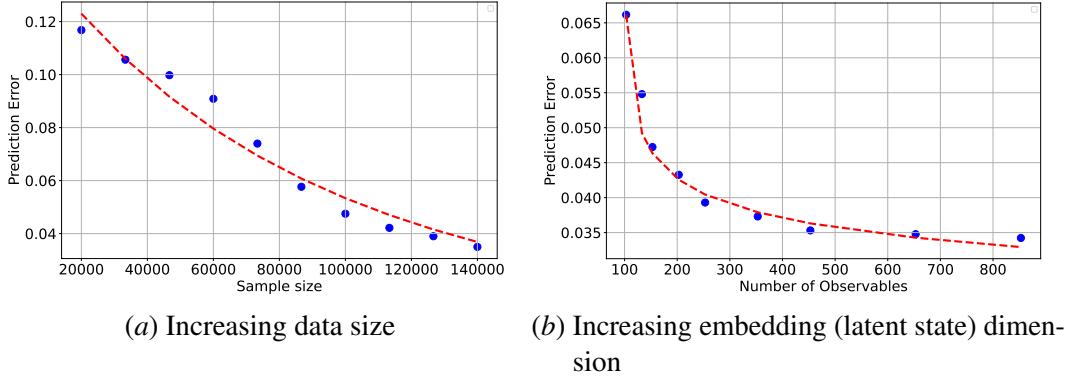


Figure 7: Validation of theorem 1.

Appendix D. Proof of Koopman operator convergence

In this section, we discuss the theoretical properties of the proposed algorithm. The convergence of data-driven Koopman operator algorithms, particularly the Extended Dynamic Mode Decomposition (EDMD) Williams et al. (2015), has been rigorously established in prior works Korda and Mezić (2018a,b). Specifically, Korda and Mezić Korda and Mezić (2018a) demonstrate that, under mild assumptions, as the sample size $m \rightarrow \infty$ and the dimension of latent state (or also called number of observables) $n \rightarrow \infty$, the estimated Koopman operator K converges to the true Koopman operator \mathcal{K} . Our theoretical analysis builds upon the results presented in Korda and Mezić (2018a,b) but differs by providing explicit convergence rates under additional assumptions. In the following sections, we focus primarily on autonomous systems without control inputs. For control systems, the results can be extended by setting $s = [x; u]^\top$.

D.1. Preliminary

Koopman Operator Theory. For clarity, we adopt terminology slightly different from that used in the main content. Here s^+ denotes the next state of s . Consider a dynamical system $s^+ = f(s)$, with $f : \mathcal{S} \rightarrow \mathcal{S}$, where \mathcal{S} is a given separable topological space. Let \mathcal{H} represent a separable Hilbert space, equipped with an inner product $\langle \cdot, \cdot \rangle$ and corresponding norm $\|\cdot\|$. The Koopman operator $\mathcal{K} : \mathcal{H} \rightarrow \mathcal{H}$ is defined as $\mathcal{K}\phi = \phi \circ f$, where $\phi \in \mathcal{H}$ are embedding functions.

Extended Dynamic Mode Decomposition (EDMD). EDMD is a data-driven method for approximating the Koopman operator \mathcal{K} within a finite-dimensional subspace. Our methodology aligns with the EDMD approach. Given a embedding function (observable) $\Phi(s) = [\phi_1(s), \phi_2(s), \dots, \phi_n(s)]^\top$,

where $\phi_i : \mathcal{S} \rightarrow \mathbb{R}$ for $i \in [1, n]$. Given m data pairs $\{(s_i, s_i^+)\}_{i=1}^m$, EDMD seeks a finite-dimensional Koopman matrix $K_{n,m} \in \mathbb{R}^{n \times n}$ that best fits the data according to the following optimization problem:

$$\min_{K \in \mathbb{R}^{n \times n}} \sum_{i=1}^m \|\Phi(s_i^+) - K\Phi(s_i)\|_2^2 \quad (9)$$

Assuming that the Gram matrix $G_{n,m}$ is invertible, the closed-form solution for the Koopman matrix is derived from:

$$K_{n,m} = A_{n,m} G_{n,m}^{-1} \quad (10)$$

$$G_{n,m} = \frac{1}{m} \sum_{i=1}^m \Phi(s_i) \Phi(s_i)^\top, \quad A_{n,m} = \frac{1}{m} \sum_{i=1}^m \Phi(s_i) \Phi(s_i^+)^\top \quad (11)$$

Denoting \mathcal{H}_n the span of $\phi_1(s), \phi_2(s), \dots, \phi_n(s)$. It is notable that any function $g \in \mathcal{H}_n$ can be represented as $g = c^\top \Phi$, where $c \in \mathbb{R}^n$.

EDMD as L_2 projection. As shown by [Korda and Mezić \(2018a\)](#), EDMD approximates the finite-dimensional Koopman operator through an L_2 projection of the true Koopman operator. We interpret these results as follows. Consider an arbitrary nonnegative measure μ on the state space \mathcal{S} , and let $L_2(\mu)$ denote the Hilbert space of all square integrable functions with respect to the measure μ . We define the $L_2(\mu)$ projection, denoted as \mathcal{P}_n^μ , of a function g onto \mathcal{H}_n :

$$\mathcal{P}_n^\mu g = \arg \min_{h \in \mathcal{H}_n} \|h - g\|_{L_2(\mu)} = \arg \min_{h \in \mathcal{H}_n} \int_{\mathcal{S}} (h - g)^2 d\mu = \Phi^\top \arg \min_{c \in \mathbb{R}^n} \int_{\mathcal{S}} (c^\top \Phi - g)^2 d\mu. \quad (12)$$

We now state a key property of $K_{n,m}$ as follows.

Theorem A.1 [Korda and Mezić \(2018a\)](#). Let $\hat{\mu}_m$ denote the empirical measure associated to the points s_1, \dots, s_m , i.e., $\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m \delta_{s_i}$, where δ_{s_i} denotes the Dirac measure at s_i . Then for any $g \in \mathcal{H}_N$,

$$K_{n,m} g = \mathcal{P}_n^{\hat{\mu}_m} \mathcal{K} g = \arg \min_{h \in \mathcal{H}_n} \|h - \mathcal{K} g\|_{L_2(\hat{\mu}_m)}, \quad \text{i.e. } K_{n,m} = \mathcal{P}_n^{\hat{\mu}_m} \mathcal{K}|_{\mathcal{H}_n}, \quad (13)$$

where $\mathcal{K}|_{\mathcal{H}_n} : \mathcal{H}_n \rightarrow \mathcal{H}$ is the restriction of the Koopman operator to the subspace \mathcal{H}_n .

The theorem states that the estimated Koopman operator $K_{n,m}$ is the L_2 projection of the true Koopman operator \mathcal{K} on the span of ϕ_1, \dots, ϕ_n with respect to the empirical measure on the samples s_1, \dots, s_m .

Consequently, the total error in approximating the true Koopman operator \mathcal{K} with its learned finite-dimensional estimate $K_{n,m}$ can be decomposed into two components: projection error and sampling error. *Sampling error:* The sampling error, denoted as $\epsilon_{\text{samp}}(n, m) = \|K_{n,m} - K_n\|$, originates from estimating the finite-dimensional Koopman operator K_n using a finite number of data samples m . *Projection error:* The projection error, denoted as $\epsilon_{\text{proj}}(n) = \|K_n \mathcal{P}_n^\mu - \mathcal{K}\|$, arises from the approximation of the infinite-dimensional Koopman operator \mathcal{K} within a finite-dimensional subspace. This conceptual framework forms the basis for the overall proof scheme presented in prior work by Korda and Mezić [Korda and Mezić \(2018a\)](#). In contrast, we extend these results by providing explicit convergence rates under certain additional assumptions.

D.2. Convergence of Sampling error

We begin by establishing the assumptions necessary for our analysis.

Assumption 1 (i.i.d samples). Data samples s_1, \dots, s_m are drawn independently from the distribution μ .

Assumption 2 (Bounded latent state). The norm of the latent state vector is bounded. Specifically, there exists a constant $B > 0$ such that $\|\Phi(s)\| \leq B$ for all $s \in \mathcal{S}$.

Assumption 3 (Invertible Gram matrix). The Gram matrix $G_{n,m}$ defined in eq. (11) is invertible, and its smallest eigenvalue is bounded below by $\gamma > 0$, such as $\lambda_{\min}(G_{n,m}) \geq \gamma$.

Assumption 1 ensures the independence of samples, which can be relaxed if the dynamical system f is ergodic and the samples s_1, \dots, s_m are taken along a trajectory of the system starting from some initial condition $s_0 \in \mathcal{S}$. Assumption 2 can be satisfied in neural network implementations by using bounded activation functions (e.g., sigmoid functions) or by constraining the network weights. While Assumption 3 requires the Gram matrix to be invertible with a bounded eigenvalue, this condition can be relaxed in practice by employing the pseudoinverse of the Gram matrix.

Under Assumption 3, the learned Koopman operator $K_{n,m}$ can be computed using eq. (10). We consider the sampling error between the learned Koopman operator $K_{n,m}$ with finite data size m , and the Koopman operator K_n obtained as the L_2 projection of the true Koopman operator onto the subspace \mathcal{H}_n with infinite data:

$$\epsilon_{\text{samp}}(n, m) = \|K_{n,m} - K_n\| = \|G_{n,m}^{-1} A_{n,m} - G_n^{-1} A_n\| \quad (14)$$

Here, $G_{n,m}$ and $A_{n,m}$ are computed using finite samples via eq. (11). While G_n and A_n , represent their expected values, equivalent to computing eq. (11) with an infinite number of samples. Under Assumption 1, by the Law of Large Numbers, $G_{n,m}$ and $A_{n,m}$ converge to the expected value G_n and A_n respectively as $m \rightarrow \infty$. Then we have:

$$G_n = \mathbb{E}[G_{n,m}] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m \Phi(s_i) \Phi(s_i)^\top\right], \quad A_n = \mathbb{E}[A_{n,m}] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m \Phi(s_i) \Phi(s_i^+)^\top\right] \quad (15)$$

We proceed to decompose the sampling error:

$$\begin{aligned} \epsilon_{\text{samp}}(n, m) &= \|A_{n,m} G_{n,m}^{-1} - A_n G_n^{-1}\| = \|A_{n,m} G_{n,m}^{-1} - A_{n,m} G_n^{-1} + A_{n,m} G_n^{-1} - A_n G_n^{-1}\| \\ &\leq \|A_{n,m}\| \cdot \|G_{n,m}^{-1} - G_n^{-1}\| + \|A_{n,m} - A_n\| \cdot \|G_n^{-1}\|. \end{aligned} \quad (16)$$

To bound each term in eq. (16), we utilize properties of matrix norms and inequalities. First, we consider the difference of the inverses of $G_{n,m}$ and G_n . Using the identity $G_{n,m}^{-1} - G_n^{-1} = G_n^{-1}(G_n - G_{n,m})G_{n,m}^{-1}$, we have:

$$\|G_{n,m}^{-1} - G_n^{-1}\| \leq \|G_n^{-1}\| \|G_n - G_{n,m}\| \|G_{n,m}^{-1}\|. \quad (17)$$

Based on the assumption 2, the spectral norm of $A_{n,m}$ can be bounded:

$$\|A_{n,m}\| \leq \frac{1}{m} \sum_{i=1}^m \|\Phi(s_i)\| \|\Phi(s_i^+)\| \leq B^2 \quad (18)$$

Similarly, under Assumption 3, the norms of inverse gram matrices are bounded

$$\|G_{n,m}^{-1}\| = \frac{1}{\lambda_{\min}(G_{n,m})} \leq \frac{1}{\gamma}, \|G_n^{-1}\| \leq \frac{1}{\gamma} \quad (19)$$

Substituting eqs. (17) to (19) back to eq. (16), we obtain:

$$\epsilon_{\text{samp}}(n, m) \leq \frac{B^2}{\gamma^2} \|G_{n,m} - G_n\| + \frac{1}{\gamma} \|A_{n,m} - A_n\| \quad (20)$$

To bound $\|G_{n,m} - G_n\|$ and $\|A_{n,m} - A_n\|$, we utilize the Matrix Bernstein Inequality [Tropp \(2012\)](#), which provides concentration bounds for sums of independent random matrices.

Matrix Bernstein Inequality. Let $\{Y_i\}_{i=1}^m$ be independent, mean-zero random matrices with dimensions $d_1 \times d_2$. Assume that each matrix satisfies $\|Y_i\| \leq L$ almost surely. Define the variance parameter

$$\sigma^2 = \left\| \sum_{i=1}^m \mathbb{E}[Y_i Y_i^\top] \right\|. \quad (21)$$

Then, for all $\epsilon \geq 0$,

$$\mathbb{P} \left(\left\| \sum_{i=1}^m Y_i \right\| \geq \epsilon \right) \leq (d_1 + d_2) \exp \left(\frac{-\epsilon^2/2}{\sigma^2 + L\epsilon/3} \right). \quad (22)$$

We apply the Matrix Bernstein Inequality to bound $\|A_{n,m} - A_n\|$. Define Y_i as:

$$Y_i = \frac{1}{m} \left(\Phi(s_i) \Phi(s_i^+)^\top - \mathbb{E}[\Phi(s_i) \Phi(s_i^+)]^\top \right). \quad (23)$$

Then $A_{n,m} - A_n = \sum_{i=1}^m Y_i$, and $E[Y_i] = 0$. Under Assumption 2, the norm of Y_i is bounded:

$$\|Y_i\| \leq \frac{2}{m} \cdot \|\Phi(s_i)\| \cdot \|\Phi(s_i^+)\| \leq \frac{2B^2}{m} \quad (24)$$

The variance parameter σ^2 is bounded as follows:

$$\sigma^2 = \left\| \sum_{i=1}^m \mathbb{E}[Y_i Y_i^\top] \right\| \leq \sum_{i=1}^m \left\| \mathbb{E}[Y_i Y_i^\top] \right\| \leq \sum_{i=1}^m \left(\frac{2B^2}{m} \right)^2 = \frac{4B^4}{m} \quad (25)$$

Applying the Matrix Bernstein Inequality with $d_1 = d_2 = n$, $L = \frac{2B^2}{m}$, $\sigma^2 = \frac{4B^4}{m}$, we obtain:

$$\mathbb{P}(\|A_{n,m} - A_n\| \geq \epsilon) = \mathbb{P} \left(\left\| \sum_{i=1}^m Y_i \right\| \geq \epsilon \right) \leq 2n \exp \left(\frac{-\epsilon^2 m/2}{4B^4 + \frac{2B^2\epsilon}{3}} \right). \quad (26)$$

Assuming $\epsilon < B^2$, and for sufficiently large m , the term $4B^4$ dominates $\frac{2B^2\epsilon}{3}$, allowing us to simplify the bound::

$$\mathbb{P}(\|A_{n,m} - A_n\| \geq \epsilon) \leq 2n \exp \left(-\frac{m\epsilon^2}{8B^4} \right). \quad (27)$$

This bound holds in probability. To ensure that the probability is at most δ , we set:

$$2n \exp\left(-\frac{m\epsilon^2}{8B^4}\right) \leq \delta. \quad (28)$$

Solving for ϵ , we find:

$$\epsilon \geq \sqrt{\frac{8B^4 \ln(2n/\delta)}{m}}. \quad (29)$$

Thus, with probability at least $1 - \delta$, the following inequality holds,

$$\|A_{n,m} - A_n\| \leq \sqrt{\frac{8B^4 \ln(2n/\delta)}{m}}. \quad (30)$$

A similar procedure applies to bound $\|G_{n,m} - G_n\|$. Using the Matrix Bernstein Inequality with probability at least $1 - \delta$,

$$\|G_{n,m} - G_n\| \leq \sqrt{\frac{8B^4 \ln(2n/\delta)}{m}}. \quad (31)$$

Substituting eqs. (30) and (31) into eq. (20), we obtain, with probability at least $(1 - \delta)^2$,

$$\epsilon_{\text{samp}}(n, m) \leq \frac{B^2}{\gamma^2} \sqrt{\frac{8B^4 \ln(2n/\delta)}{m}} + \frac{1}{\gamma} \sqrt{\frac{8B^4 \ln(2n/\delta)}{m}} = \frac{2B^2 \sqrt{2 \ln(2n/\delta)}}{\gamma \sqrt{m}} \left(\frac{B^2}{\gamma} + 1 \right) \quad (32)$$

For simplicity, we mainly consider the convergence rate related to m, n , and then with a high probability,

$$\epsilon_{\text{samp}}(n, m) \leq \mathcal{O}\left(\sqrt{\frac{\ln(n)}{m}}\right) \quad (33)$$

For larger sample sizes m , such that $m = \mathcal{O}(n \ln(n))$, with high probability we have:

$$\epsilon_{\text{samp}}(n, m)|_{m=\mathcal{O}(n \ln(n))} \leq \mathcal{O}\left(\frac{1}{\sqrt{n}}\right) \quad (34)$$

This result indicates that, under the given assumptions, the sampling error decreases inversely with the square root of the number of observables n , when the number of samples m grows proportionally to $n \ln n$.

D.3. Convergence of Projection Error

In this section, we establish the convergence of the projection error $\epsilon_{\text{proj}}(n)$ as the dimension n of the subspace \mathcal{H}_n increases. We begin by introducing additional assumptions necessary for our analysis.

Assumption 4 (Orthogonal basis). *The embedding functions ϕ_1, \dots, ϕ_n are orthogonal (independent). i.e. $\langle \phi_i, \phi_j \rangle = \delta_{ij}$, for all $i, j = 1, \dots, n$, where δ_{ij} is the Kronecker delta.*

Assumption 5 (Bounded Koopman Operator). *The Koopman operator $\mathcal{K} : \mathcal{H} \rightarrow \mathcal{H}$ is bounded, i.e. $\|\mathcal{K}\| \leq M < \infty$.*

Assumption 4 is nonrestrictive since any countable dense subset of \mathcal{H} can be orthonormalized using the Gram–Schmidt process. Assumption 5 holds, for instance, when the system dynamics are Lipschitz continuous and the state space is bounded.

Given a function $\psi \in \mathcal{H}$, we define the projection error as

$$\epsilon_{\text{proj}}(n) = \|K_n \mathcal{P}_n^\mu \psi - \mathcal{K}\psi\| = \int_S \|K_n \mathcal{P}_n^\mu \psi - \mathcal{K}\psi\| d\mu \quad (35)$$

where K_n is the finite-dimensional approximation of the Koopman operator defined on \mathcal{H}_n , and \mathcal{P}_n^μ is the orthogonal projection onto \mathcal{H}_n .

We decompose ψ into its projection onto \mathcal{H}_n and its orthogonal complement: $\psi = \mathcal{P}_n^\mu \psi + (I - \mathcal{P}_n^\mu)\psi$, where I is the identity operator. Then we have:

$$\epsilon_{\text{proj}}(n) = \|K_n \mathcal{P}_n^\mu \psi - \mathcal{K}\psi\| \quad (36)$$

$$= \|K_n \mathcal{P}_n^\mu \mathcal{P}_n^\mu \psi - \mathcal{K}\mathcal{P}_n^\mu \psi + K_n \mathcal{P}_n^\mu (I - \mathcal{P}_n^\mu)\psi - \mathcal{K}(I - \mathcal{P}_n^\mu)\psi\| \quad (37)$$

According to the definition of the orthogonal projection \mathcal{P}_n^μ , we have $\mathcal{P}_n^\mu \mathcal{P}_n^\mu \psi = \mathcal{P}_n^\mu \psi$, $\mathcal{P}_n^\mu \cdot (I - \mathcal{P}_n^\mu)\psi = 0$. Recall from Theorem A.1 that $K_n = \mathcal{P}_n^\mu \mathcal{K}$. Therefore, we can rewrite eq. (37) as :

$$\epsilon_{\text{proj}}(n) = \|K_n \mathcal{P}_n^\mu \psi - \mathcal{K}\mathcal{P}_n^\mu \psi - \mathcal{K}(I - \mathcal{P}_n^\mu)\psi\| \quad (38)$$

$$= \|(\mathcal{P}_n^\mu - I)\mathcal{K}\mathcal{P}_n^\mu \psi + \mathcal{K}(\mathcal{P}_n^\mu - I)\psi\| \quad (39)$$

$$\leq \|(\mathcal{P}_n^\mu - I)\| \|\mathcal{K}\mathcal{P}_n^\mu \psi\| + \|\mathcal{K}\| \|(\mathcal{P}_n^\mu - I)\| \|\psi\| \quad (40)$$

Under Assumption 5, $\|\mathcal{K}\| \leq M$. Under assumption 2, $\|\phi_i\| \leq B$. Let $\psi = \sum_{i=1}^{\infty} c_i \phi_i$. By Parseval's identity $\sum_{i=1}^{\infty} |c_i|^2 = 1$. Then $\|\psi\| \leq B$. Therefore, we can rewrite eq. (40) as :

$$\epsilon_{\text{proj}}(n) \leq 2MB \|(\mathcal{P}_n^\mu - I)\| \quad (41)$$

From Equation (41), we observe that the projection error depends on the norm of the residual $(\mathcal{P}_n^\mu - I)\psi$. As $n \rightarrow \infty$, the subspace \mathcal{H}_n becomes dense in \mathcal{H} , and \mathcal{P}_n^μ converges strongly to the identity operator I [Korda and Mezić \(2018a\)](#). Therefore,

$$\lim_{n \rightarrow \infty} \epsilon_{\text{proj}}(n) = 0, \quad \lim_{n \rightarrow \infty} K_n \mathcal{P}_n^\mu = \mathcal{K} \quad (42)$$

Recall that K_n is an operator on the finite-dimensional subspace \mathcal{H}_n spanned by the embedding functions ϕ_1, \dots, ϕ_n . And \mathcal{K} is an operator on \mathcal{H} . The convergence of K_n to \mathcal{K} implies convergence of their eigenvalues and eigenfunctions under certain conditions. Prior work [Korda and Mezić \(2018a\)](#) shows the convergence of eigenvalue as the following theorem.

Theorem A.2 [Korda and Mezić \(2018a\)](#). *If assumption 4 and assumption 5 hold, and λ_n is a sequence of eigenvalue of K_n with associated normalized eigenfunctions $\phi_n \in \mathcal{H}_n$, then there exists a subsequence $(\lambda_{n_i}, \phi_{n_i})$ such that*

$$\lim_{i \rightarrow \infty} \lambda_{n_i} = \lambda^*, \quad \phi_{n_i} \xrightarrow{\text{weak convergence}} \phi^* \quad (43)$$

where $\lambda^* \in \mathbb{C}$ and $\phi^* \in \mathcal{H}$ are eigenvalue and eigenfunction of Koopman operator, such that $\mathcal{K}\phi^* = \lambda^*\phi^*$.

This theorem demonstrates the convergence of eigenvalues and eigenfunctions of K_n to those of \mathcal{K} as $n \rightarrow \infty$. Let the sorted (decreasing order) eigenvalues of the Koopman operator \mathcal{K} are $\lambda_1^*, \lambda_2^*, \dots$, and corresponding eigenfunctions are $\phi_1^*, \phi_2^*, \dots$. The sorted eigenvalues of the K_n are $\lambda_1, \lambda_2, \dots, \lambda_n$, and corresponding eigenfunctions are $\phi_1, \phi_2, \dots, \phi_n$. To provide an explicit convergence rate for the projection error, we introduce further assumptions.

Assumption 6 (Spectral Properties of \mathcal{K}). *The eigenvalues of the Koopman operator \mathcal{K} decay sufficiently fast, for instance, $|\lambda_i| \leq \frac{C}{i^\alpha}$ for some constant $C > 0$, $\alpha > \frac{1}{2}$.*

Assumption 7 (Properly learned eigenfunctions). *The embedding functions $\Phi(\cdot) = [\phi_1, \dots, \phi_n]^\top$ correspond to the first n eigenfunctions of \mathcal{K} associated with the largest eigenvalues in magnitude. That is, $\lambda_i = \lambda_i^*$ or $\langle \lambda_i \phi_i, f \rangle = \langle \lambda_i^* \phi_i^*, \mathcal{P}_n^\mu f \rangle$ for $f \in \mathcal{H}$.*

Assumption 6 holds for systems where the Koopman operator has rapidly decaying spectral components, such as stable linear systems. Assumption 7 implies that the learned embedding functions capture the most significant modes of the system, which is reasonable given the universal approximation capabilities of neural networks and the optimization objective in Equation (9).

Under assumption 7, we can quantify the convergence rate of $\epsilon_{\text{proj}}(n)$. Let $\psi \in \mathcal{H}$, be expressed in terms of the eigenfunctions of \mathcal{K} : $\psi = \sum_{i=1}^{\infty} c_i \phi_i^* \in \mathcal{H}$ with $\sum_{i=1}^{\infty} |c_i|^2 = 1$. The projection error is given by

$$\epsilon_{\text{proj}}(n) = \|K_n \mathcal{P}_n^\mu \psi - \mathcal{K}\psi\| = \left\| \sum_{i=1}^{\infty} c_i K_n \mathcal{P}_n^\mu \phi_i^* - \sum_{i=1}^{\infty} c_i \mathcal{K} \phi_i^* \right\| \quad (44)$$

$$= \left\| \sum_{i=1}^n c_i \lambda_i \phi_i^* - \sum_{i=1}^{\infty} c_i \lambda_i^* \phi_i^* \right\| = \left\| \sum_{i=1}^n c_i (\lambda_i - \lambda_i^*) \phi_i^* + \sum_{i=n+1}^{\infty} c_i \lambda_i^* \phi_i^* \right\| \quad (45)$$

$$= \left\| \sum_{i=n+1}^{\infty} c_i \lambda_i^* \phi_i^* \right\| \leq B \cdot \left(\sum_{i=n+1}^{\infty} |\lambda_i^*|^2 \right)^{1/2} \quad (46)$$

Considering assumption 6, we can estimate the projection error

$$\epsilon_{\text{proj}}(n) \leq B \left(\sum_{i=n+1}^{\infty} \left(\frac{C}{i^\alpha} \right)^2 \right)^{1/2} \leq BC \left(\int_n^{\infty} \frac{dx}{x^{2\alpha}} \right)^{1/2} \quad (47)$$

$$= BC \left(\frac{1}{(2\alpha - 1)n^{2\alpha-1}} \right)^{1/2} = \frac{BC}{\sqrt{2\alpha - 1}} \cdot \frac{1}{n^{\alpha-\frac{1}{2}}}. \quad (48)$$

Therefore, we have the convergence rate of projection error:

$$\epsilon_{\text{proj}}(n) \leq \mathcal{O} \left(\frac{1}{n^{\alpha-\frac{1}{2}}} \right) \quad (49)$$

For $\alpha = 1$, we have:

$$\epsilon_{\text{proj}}(n) \leq \mathcal{O} \left(\frac{1}{\sqrt{n}} \right) \quad (50)$$

Thus, under the given assumptions, the projection error decreases inversely with the square root of n .

D.4. Convergence of Koopman Operator Under Incremental Strategy

By combining the convergence results of the sampling error eq. (34) and the projection erroreq. (42), we conclude that the estimated Koopman operator $K_{m,n}$ converges to the true Koopman operator \mathcal{K} . This convergence has also been established in previous work by Korda and Mezić [Korda and Mezić \(2018a\)](#).

Theorem A.3 *Under the assumptions of 1) data samples s_1, \dots, s_m are i.i.d distributed; 2) the latent state is bounded, $\|\phi(s)\| < \infty$; 3) the embedding functions ϕ_1, \dots, ϕ_n are orthogonal (independent). The learned Koopman operator $K_{m,n}$ converges to true Koopman Operator:*

$$\lim_{m=\Omega(n \ln(n)), n \rightarrow \infty} K_{m,n} \rightarrow \mathcal{K} \quad (51)$$

Unlike prior work, we provide explicit convergence rates for both the sampling error and the projection error eqs. (34) and (49), leading to an overall error bound for $K_{m,n}$.

Theorem A.4 *Under the assumptions of 1) data samples s_1, \dots, s_m are i.i.d distributed; 2) the latent state is bounded, $\|\phi(s)\| < \infty$; 3) the embedding functions ϕ_1, \dots, ϕ_n are orthogonal (independent).*

(a) **Sampling Error Rate:** The sampling error decreases with the number of samples m as:

$$\epsilon_{\text{samp}}(n, m) \leq \mathcal{O}\left(\sqrt{\frac{\ln(n)}{m}}\right) \quad (52)$$

(b) **Projection Error Rate:** Assuming additional conditions: 4) The eigenvalues of the Koopman operator \mathcal{K} decay sufficiently fast, for instance, $|\lambda_i| \leq \frac{C}{i^\alpha}$ for some constant $C > 0, \alpha > \frac{1}{2}$. 5) The embedding functions $\Phi(\cdot) = [\phi_1, \dots, \phi_n]^\top$ correspond to the first n eigenfunctions of \mathcal{K} associated with the largest eigenvalues in magnitude. Then, the projection error decreases with n :

$$\epsilon_{\text{proj}}(n) \leq \mathcal{O}\left(\frac{1}{n^{\alpha-\frac{1}{2}}}\right) \quad (53)$$

(c) **Overall Error Bound:** The total approximation error between $K_{m,n}$ and true koopman operator \mathcal{K} satisfies:

$$\text{error} \leq \mathcal{O}\left(\sqrt{\frac{\ln(n)}{m}}\right) + \mathcal{O}\left(\frac{1}{n^{\alpha-\frac{1}{2}}}\right) \quad (54)$$

We conducted experiments to evaluate the validity of Theorem A.4; the results are presented in Appendix C.2.

Appendix E. Expeiment Details

E.1. Task Settings

We summarize our test suites in Table 3, and present the tracking failure threshold ϵ_{fail} in Table 4. It is worth noting that robots performing different tasks have distinct physical structures and movement patterns, necessitating different ϵ_{fail} values. These hyperparameters are selected when the robot is on the verge of falling and entering an unrecoverable state.

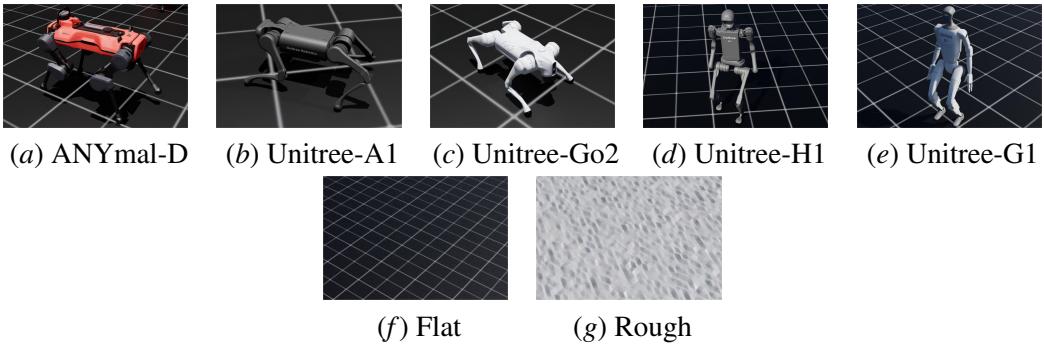


Figure 8: Legged robots and terrain in our test suites.

Table 3: The settings of test suites environment for our experiments

Task Settings	Producer		Terrain		Robot Type	
	Unitree	ANYmal	Flat	Rough	Quadruped	Humanoid
IsaacLab	Anymal-D (\mathbb{R}^{12})		✓	✓		✓
	A1 (\mathbb{R}^{12})	✓		✓		✓
	Go2 (\mathbb{R}^{12})	✓		✓	✓	
Robot	H1 (\mathbb{R}^{19})	✓		✓		✓
	G1 (\mathbb{R}^{23})	✓		✓		✓

E.2. Dataset Settings

All five legged robots are trained on IsaacLab [Mittal et al. \(2023\)](#). The initial dataset $\mathcal{D}^{(0)}$ consists of 6e4 trajectories of length $l_{\mathcal{D}^{(0)}}$, and the reference repository \mathcal{R} , containing 3e3 demonstrations of length $l_{\mathcal{R}}$ showcasing diverse and rich walking styles.

To achieve a more uniform data distribution under specific gait and contact modes, we apply random initialization techniques for constructing $\mathcal{D}^{(0)}$ and \mathcal{R} . First, we randomize the initial movement commands, including heading direction, x-axis linear velocity, and y-axis linear velocity in world coordinates, where the x-axis aligns with the heading direction and the y-axis is perpendicular to it. After collecting 6e4 trajectories of length l_{init} (where $l_{init} > l_{\mathcal{D}^{(0)}}$), we randomly extract continuous segments of length $l_{\mathcal{D}^{(0)}}$ to enrich the robot's range of motion and postures. All detailed hyperparameters are provided in Table 5.

In the following lifting phase, incremental dataset \mathcal{D}_{incre} , containing 3e4 trajectories of length $l_{\mathcal{D}^{(0)}}$, will be added to the training dataset.

E.3. Algorithm Settings

The detailed dimensional information for the original and latent spaces for each task and algorithm is provided in Table 5. For simplicity, we organize the hyperparameters corresponding to each task in an array format as follows: (Flat-Anymal-D, Flat-Unitree-A1, Flat-Unitree-Go2, Rough-Unitree-Go2, Flat-Unitree-H1, Flat-Unitree-G1, Rough-Unitree-G1).

Table 4: The settings of hyperparameter ϵ_{fail} for test suites

ϵ_{fail}	Settings	Anymal-D	A1	Go2	H1	G1
Terrain Type	Flat	0.18	0.16	0.16	0.15	0.10
	Rough	-	-	0.12	-	0.08

We set a fixed step size Δn for dimension increment at 100. The horizon length H for loss computation and MPC solving is set to 24 for the Unitree-H1 robot and 16 for the others.

All networks implemented for testing the algorithms are Residual Neural Networks (He et al. (2015)), featuring residual blocks structured as {Linear}–{Relu}–{Linear}–{Residual}–{RELU}. The hidden dimensions and number of blocks are listed in Table 5. All networks are trained using the Adam optimizer with a Cosine Annealing learning rate scheduler. The initial learning rate and training epochs are 1e-3 and 100, respectively.

We normalize data in all test suites to mitigate the misaligned scale for each physical variable. The initial trajectories $m_{\mathcal{D}^{(0)}}$ is set to 6e6 with initial trajectories length $l_{\mathcal{D}^{(0)}}$ 100 for flat terrain and 200 for rough terrain. The clipped trajectory length l_{init} is equal with horizon length H . Then we construct reference repository \mathcal{R} with reference number $m_{\mathcal{R}}$ 3000 and reference length 500.

Other unique hyperparameters for each algorithm follow the original paper to attain best performance.

Each model is trained on a server with a 48-core Intel(R) Xeon(R) Silver 6426Y CPU @ 2.5.GHz, four Nvidia RTX A6000 GPU with 48GB memory, and Ubuntu 22.04.

E.4. Definition of Tracking Metrics

- Joint-relative mean per-joint position error (E_{JrPE}): $\frac{1}{200*J} \sum_{t=1}^{200} \|j_t - j_t^*\|_1$, where j_t is the measured DoF position, j_t^* is the reference DoF position, J is the number of DoF.
- Joint-relative mean per-joint velocity error (E_{JrVE}): $\frac{1}{200*J} \sum_{t=1}^{200} \|\dot{j}_t - \dot{j}_t^*\|_1$
- Joint-relative mean per-joint acceleration error (E_{JrAE}): $\frac{1}{200*J} \sum_{t=1}^{200} \|\ddot{j}_t - \ddot{j}_t^*\|_1$
- Root mean position error (E_{RPE}): $\frac{1}{200*3} \sum_{t=1}^{200} \|p_t - p_t^*\|_1$, where p_t is the measured root position, p_t^* is the reference root position.
- Root mean linear velocity error (E_{RLVE}): $\frac{1}{200*3} \sum_{t=1}^{200} \|\dot{p}_t - \dot{p}_t^*\|_1$
- Root mean orientation error (E_{ROE}): $\frac{1}{200*4} \sum_{t=1}^{200} \|r_t - r_t^*\|_1$, where r_t is the measured root orientation (represented as a quaternion), r_t^* is the reference root orientation (represented as a quaternion).
- Root mean angular velocity error (E_{RLAE}): $\frac{1}{200*3} \sum_{t=1}^{200} \|\dot{r}_t - \dot{r}_t^*\|_1$

E.5. Tracking Metrics for Each Task

Table 5: Important hyperparameters of different algorithms in our experiments

Policy Parameter		Ours	DKRL	DKUC	DKAC	NNDM
Original space dimension	n'	(35, 35, 35, 35, 45, 53, 53)	(35, 35, 35, 45, 53, 53)	(35, 35, 35, 45, 53, 53)	(35, 35, 35, 45, 53, 53)	(35, 35, 35, 45, 53, 53)
Control input dimension	m'	(12,12,12,12,19,23,23)	(12,12,12,12,19,23,23)	(12,12,12,12,19,23,23)	(12,12,12,12,19,23,23)	(12,12,12,12,19,23,23)
Initial latent space dimension	$n^{(0)}$	(512, 384, 384, 535, 214, 222, 453)	(512, 384, 384, 535, 214, 222, 453)	(512, 384, 384, 535, 214, 222, 453)	(512, 384, 384, 535, 214, 222, 453)	(512, 384, 384, 535, 214, 222, 453)
Fixed step size	Δn	100	100	100	100	100
Inference horizon	H	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)
Initial training epochs	$J^{(0)}$	100	100	100	100	100
Initial learning rate		1e-3	1e-3	1e-3	1e-3	1e-3
Network optimizer		Adam	Adam	Adam	Adam	Adam
Network scheduler		CosineAnnealnLR	CosineAnnealnLR	CosineAnnealnLR	CosineAnnealnLR	CosineAnnealnLR
Network hidden dimension		(256, 256, 256, 256, 256, 256)	(256, 256, 256, 256, 256, 256)	(256, 256, 256, 256, 256, 256)	(256, 256, 256, 256, 256, 256)	(256, 256, 256, 256, 256, 512)
Network blocks number		(3,3,3,3,3,3)	(3,3,3,3,3,3)	(3,3,3,3,3,3)	(3,3,3,3,3,3)	(3,3,3,3,3,3)
Discount Factor	γ	0.99	0.99	0.99	0.99	0.99
Loss Weight	α	0.1	0.1	0.1	0.1	0.1
Data normalization		✓	✓	✓	✓	✓
Initial trajectory length	l_{init}	(100, 100, 100, 200, 100, 100, 200)	(100, 100, 100, 200, 100, 100, 200)	(100, 100, 100, 200, 100, 100, 200)	(100, 100, 100, 200, 100, 100, 200)	(100, 100, 100, 200, 100, 100, 200)
Clipped trajectory length	$l_{D^{(0)}}$	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)	(16,16,16,16,24,16,16)
Reference trajectory length	l_R	(500, 500, 500, 500, 500, 500, 500)	(500, 500, 500, 500, 500, 500, 500)	(500, 500, 500, 500, 500, 500, 500)	(500, 500, 500, 500, 500, 500, 500)	(500, 500, 500, 500, 500, 500, 500)
GMM cluster number		5	5	5	5	5

Algorithm	General Tracking Metrics							
	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0467	0.9227	55.7386	0.1464	0.0418	0.1322	0.3367	194.5080
DKRL	0.0794	1.2533	69.3621	0.2752	0.0549	0.2077	0.4619	182.3020
DKAC	0.2422	1.7859	69.4619	0.5077	0.2532	0.3533	0.7922	44.6020
DKUC	0.3137	1.4798	47.1150	0.5221	0.3251	0.3472	0.7125	27.9680
NNMD	0.1841	1.8721	75.2555	0.5357	0.2126	0.3564	0.7197	35.0360

Table 6: The average tracking metrics evaluated for each algorithm on Flat-Anymal-D.

Algorithm	General Tracking Metrics							
	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0322	0.8212	59.6711	0.0930	0.0256	0.0800	0.2657	195.4880
DKRL	0.0498	1.0385	71.4132	0.1180	0.0401	0.1039	0.3671	161.4340
DKAC	0.1462	3.8430	276.9420	0.3574	0.1101	0.2874	1.3771	10.8920
DKUC	0.0382	0.9525	69.6217	0.0997	0.0287	0.0896	0.3132	176.9560
NNMD	0.2166	3.9797	253.8551	0.4614	0.1215	0.3371	1.1712	3.8280

Table 7: The average tracking metrics evaluated for each algorithm on Flat-Unitree-A1.

Algorithm	General Tracking Metrics							
	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0428	0.9563	67.1742	0.1127	0.0364	0.0934	0.2946	195.1240
DKRL	0.0734	1.5231	104.0327	0.1817	0.0557	0.1432	0.4822	159.8200
DKAC	0.1710	2.9952	186.1956	0.3994	0.1956	0.2931	1.0813	19.5120
DKUC	0.1328	1.6218	89.0778	0.2163	0.1116	0.1814	0.5298	115.7020
NNMD	0.1716	3.3656	203.0976	0.4645	0.1290	0.3181	1.0053	4.6600

Table 8: The average tracking metrics evaluated for each algorithm on Flat-Unitree-Go2.

General Tracking Metrics								
Algorithm	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0425	0.7963	55.5985	0.1001	0.1227	0.1021	0.4320	191.7280
DKRL	0.0846	1.5861	116.1124	0.1843	0.2604	0.1554	0.6729	71.1740
DKAC	0.1648	1.7733	91.0318	0.3195	0.4493	0.2015	0.7549	21.8420
DKUC	0.1177	0.8725	35.6163	0.1687	0.2227	0.1492	0.5043	82.0040
NNDM	0.1180	2.2806	147.1042	0.3023	0.4345	0.1997	0.8586	28.0060

Table 9: The average tracking metrics evaluated for each algorithm on Rough-Unitree-Go2.

General Tracking Metrics								
Algorithm	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0377	0.2983	15.1954	0.1860	0.0874	0.1949	0.3396	181.4740
DKRL	0.0992	0.5034	17.7740	0.3657	0.2438	0.2538	0.5304	98.7680
DKAC	0.1951	0.9522	32.7991	0.3813	0.3077	0.2613	0.7258	40.0660
DKUC	0.1925	0.9012	32.0362	0.3701	0.2438	0.2563	0.6709	71.2460
NNDM	0.1105	0.8381	31.3497	0.4377	0.2752	0.2792	0.5807	109.2700

Table 10: The average tracking metrics evaluated for each algorithm on Flat-Unitree-H1.

General Tracking Metrics								
Algorithm	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0065	0.2346	17.1064	0.0219	0.0126	0.0365	0.1174	198.1140
DKRL	0.0858	1.1325	55.2168	0.5004	0.2182	0.3082	0.6091	73.2240
DKAC	0.1819	2.0094	110.0289	0.3974	0.3332	0.3274	0.9012	14.0400
DKUC	0.1579	0.9549	44.7592	0.3077	0.2420	0.2746	0.6048	51.1280
NNDM	0.1158	1.5910	87.3439	0.4060	0.2896	0.2994	0.7254	37.4180

Table 11: The average tracking metrics evaluated for each algorithm on Flat-Unitree-G1.

Algorithm	General Tracking Metrics							
	Joint-relative ↓			Root-relative ↓				Survival ↑
	E_{JrPE}	E_{JrVE}	E_{JrAE}	E_{RPE}	E_{ROE}	E_{RLVE}	E_{RAVE}	T_{Sur}
Ours	0.0351	0.5202	31.5414	0.2017	0.1408	0.2120	0.5166	162.7240
DKRL	0.1039	0.8387	48.7527	0.4591	0.2198	0.2903	0.8203	71.9560
DKAC	0.1699	1.1268	56.4011	0.4061	0.2754	0.2979	0.7680	24.2240
DKUC	0.1503	0.7970	35.7952	0.3692	0.2186	0.2779	0.5559	52.2300
NNDM	0.0905	1.4870	94.1604	0.4262	0.2918	0.3075	0.9143	30.0780

Table 12: The average tracking metrics evaluated for each algorithm on Rough-Unitree-G1.

