Sample-Efficient Online Control Policy Learning with Real-Time Recursive Model Updates

Zixin Zhang, James Avtges, Todd D. Murphey
Department of Mechanical Engineering
Northwestern University, United States
{zixinz, javtges}@u.northwestern.edu, t-murphey@northwestern.edu

Abstract:

Data-driven control methods need to be sample-efficient and lightweight, especially when data acquisition and computational resources are limited—such as during learning on hardware. Most modern data-driven methods require large datasets and struggle with real-time updates of models, limiting their performance in dynamic environments. Koopman theory formally represents nonlinear systems as linear models over observables, and Koopman representations can be determined from data in an optimization-friendly setting with potentially rapid model updates. In this paper, we present a highly sample-efficient, Koopman-based learning pipeline: Recursive Koopman Learning (RKL). We identify sufficient conditions for model convergence and provide formal algorithmic analysis supporting our claim that RKL is lightweight and fast, with complexity independent of dataset size. We validate our method on a simulated planar two-link arm and a hybrid nonlinear hardware system with soft actuators, showing that real-time recursive Koopman model updates improve the sample efficiency and stability of data-driven controller synthesis—requiring only $<10\%$ of the data compared to benchmarks. The high-performance C++ codebase is open-sourced. Website: https://www.zixinatom990.com/home/robotics/corl-2025-recursive-koopman-learning.

Keywords: Koopman Operator, Control

1 Introduction

Designing controllers for nonlinear, dynamic robotic systems is challenging. Conventional model-based approaches rely on first principles to derive system models and then compute control inputs by solving constrained optimal control problems, often through trajectory optimization. While these methods are mathematically rigorous and have been successfully deployed on robotic systems [1, 2, 3, 4], they exhibit inherent limitations: they are difficult to apply to systems that cannot be easily modeled using first principles, such as soft robots; they require significant time and effort to be adapted for highly dynamic

Figure 1: Recursive Koopman Learning Pipeline

tasks or complex environments with significant uncertainties; they struggle with designing controllers for hybrid dynamical systems, especially those involving multiple contact modes; and as the number of constraints increases, these methods impose escalating demands on computational resources and solver performance.
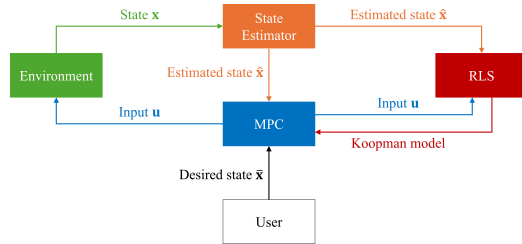
Data-driven approaches have advantages in dealing with uncertainties and complex dynamics. Over the past decade, these approaches—especially Reinforcement Learning (RL)—have made significant advances. RL algorithms [5, 6, 7] have been widely used to design controllers for various robotic systems and have achieved remarkable performance in challenging tasks, including soft robot control [8], dexterous manipulation [9, 10], vehicle control [11], and dynamic locomotion [12, 13]. However, RL typically require large amounts of data and have low sample efficiency, and their training demands substantial computational resources. These factors make it difficult to perform RL training solely on hardware—an important consideration for systems that cannot be simulated with high fidelity, such as soft robots. Additionally, due to the stochastic processes involved and the use of neural networks, RL often lack convergence guarantees and mathematical interpretability.

Given the disadvantages of conventional model-based control and RL, there is a need for sample-efficient and computationally lightweight data-driven control policy learning methods. A natural idea is to use linear model-based methods—linearity brings convexity and makes computations lighter and faster, and model-based methods tend to have higher sample efficiency. Standard linearization—which is performed locally and in the original state space—can only produce accurate local models. In contrast, Koopman theory can help us find *exact linear models for nonlinear systems* in infinite dimensional spaces. With advances in computational capabilities, it has become feasible to estimate Koopman operators effectively using data-driven methods such as Dynamic Mode Decomposition (DMD) and Extended Dynamic Mode Decomposition (EDMD) [14, 15]. As a result, Koopman theory has been successfully applied in various fields, including fluid dynamics [16], physics [17], power systems [18], and robotics [19, 20, 21, 22, 23].

To improve sample efficiency, a key strategy is to identify and use the most informative data. A widely accepted but rarely formalized hypothesis—referred to here as the Attempting Control Goal (ACG) hypothesis—suggests that data collected while attempting a control objective with an ideal policy is particularly informative for that task. An ideal policy is a control policy that is dynamically feasible and globally optimizes a specified performance criterion—such as minimizing a cost function or maximizing reward—under the true system dynamics. Although such a policy is unattainable, researchers aim to approximate it and use the resulting demonstration data to train control policies. In imitation learning [24, 25], for example, human or animal instructors are treated as noisy surrogates of the ideal policy, and their successful demonstrations are used for training.

However, certain systems are so challenging that even skilled humans struggle to provide consistently successful demonstrations, and the additional resources required to collect these data cannot be ignored. This motivates a weak ACG hypothesis: when a high-quality approximation of the ideal policy is unavailable, does the iterative loop of (i) computing the best policy attainable with the data so far and (ii) repeatedly attempting to control the system still yield data that are highly informative? This motivates us to adopt real-time online model updates.

Researchers have attempted updating NN models with online data [11, 26], but these model-updating methods are heavy, slow, far from achieving real-time performance, and sample-inefficient. Linear models are particularly well-suited for this purpose, and researchers have explored updating Koopman models with new data. While a few papers mention model updates [21, 23], they primarily rely on retraining or iterative numerical solvers to update the parameters, which can be computationally intensive and slow. Additionally, some works have proposed faster online updating algorithms for linear models inspired by the Recursive Least Squares (RLS) algorithm [25, 27, 28]; however, these approaches lack comprehensive convergence and complexity analyses, and no experimental validation has been conducted on real-world robotic systems. [28] also uses RLS and MPC, but its update termination condition—which hinders convergence analysis—and MPC operate on the state space rather than the observable, questionable for Koopman-based methods.

Several studies have analyzed the convergence properties of EDMD. For instance, [29, 30, 31] have provided detailed discussions on the convergence of EDMD for both stochastic and deterministic systems, under i.i.d. and ergodic sampling assumptions. However, no existing work formally analyzes the convergence of EDMD on Markov chains—a key property of data generated by dynamical systems. The original EDMD paper [15] conceptually discusses approximations of the stochastic

Koopman operator for Markov processes, but does not present a formal analysis. In addition, it omits the assumptions necessary to apply the Strong Law of Large Numbers for Markov chains [32], and its treatment of the sampling density does not impose any assumptions on dataset structure. [29] derives probabilistic error bounds on finite data assuming *i.i.d.* or ergodic sampling with the exponentially stable Koopman semigroup—a strong assumption which presupposes all non-constant observables decay exponentially toward their equilibrium values over time—without considering data correlation that is critical when using data from trajectories.

In this paper, we introduce Recursive Koopman Learning (RKL), a sample-efficient, Koopman-based learning pipeline. By leveraging Koopman theory, EDMD [15], RLS [25, 27, 28], and Model Predictive Control (MPC), RKL enables rapid control policy learning and real-time recursive model updates. RKL significantly outperforms other data-driven methods in sample efficiency. On a simulated planar two-link arm, it surpasses RL benchmarks using <5% of their data and with far lower computational cost. In hardware, RKL requires only 1 minute and 20 seconds (8,000 steps, 6,000 pretrained) to design a high-performance controller (Fig. 2) for a highly nonlinear, hybrid-dynamic soft robot (Appx. Fig. 4). In contrast, a SotA RL algorithm takes 2 hours and 46 minutes (100,000 steps) and only attains <50% of RKL's performance. We also provide the first formal convergence analysis of EDMD and RLS under continuous data growth within Markov chains, establishing sufficient conditions for convergence—crucial for practical use and previously unaddressed. We also discuss how to relate these conditions to the ACG hypothesis and our experimental results. Additionally, we present detailed algorithmic analyses of EDMD and RLS, demonstrating that RKL is computationally lightweight, with complexity independent of dataset size.

The contributions of the paper are summarized as follows:
- We present RKL, an extremely sample-efficient, data-driven control policy learning pipeline with low computational cost, supported by rigorous convergence and algorithmic complexity analysis as well as thorough experiments on hardware and in simulation.
- We explicitly identify the sufficient conditions for the convergence of EDMD and RLS under continuous data growth in the context of Markov chains, addressing a gap in prior research. Additionally, we discuss the association between this analysis and the ACG hypothesis, and how they can guide the design of sample-efficient model learning methods.
- We provide a high-performance, multi-threaded C++ codebase designed to enable easy deployment and reproducibility: https://github.com/zixinz990/recursive-koopman-learning.git.

This paper is organized as follows: Sec. 2 provides the necessary background. Sec. 3 details the RKL pipeline. Sec. 4 presents our experimental results and discuss their relation to the ACG hypothesis and our convergence analysis. Sec. 5 summarizes our conclusions. Finally, Sec. 6 discusses the limitations of our method and outlines directions for future research. The appendix is in the supplementary materials.

2 Background

This section briefly presents the mathematical background of the proposed method, including Koopman theory, EDMD, RLS and MPC.

2.1 Koopman Theory

Consider a discrete-time dynamical system $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$, $\mathbf{x} \in \mathbf{X} \subset \mathbb{R}^{n_x}$. Define an observation function (or "observable") $\mathbf{z} := \boldsymbol{\phi}(\mathbf{x}) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}, n_z \geq n_x$, which lifts the state into a higher-dimensional space. Let $\mathbf{H}$ be a Hilbert space on $\mathbf{X}$. If the components of the observation function $\{\phi_1, \phi_2, ...\}$ form an orthonormal set of basis functions spanning $\mathbf{H}$, and the composition function $\boldsymbol{\phi} \circ \mathbf{f}$ is also an element of the Hilbert space, then there exists a linear operator $\mathbf{K}$ such that $\boldsymbol{\phi} \circ \mathbf{f}(\mathbf{x}) = \mathbf{K}\boldsymbol{\phi}(\mathbf{x})$, when $n_z = \infty$ [33, 34]. We also need consider control inputs $\mathbf{u} \in \mathbb{R}^{n_u}$ to design controllers. Define another observation function $\mathbf{g} := \boldsymbol{\psi}(\mathbf{u}) : \mathbb{R}^{n_u} \to \mathbb{R}^{n_g}, n_g \geq n_u$ and a new pair of variables $\boldsymbol{\alpha}_k \in \mathbb{R}^{n_z+n_g} = [\boldsymbol{\phi}(\mathbf{x}_k), \boldsymbol{\psi}(\mathbf{u}_k)]$ and $\boldsymbol{\beta}_k \in \mathbb{R}^{n_z+n_g} = [\boldsymbol{\phi}(\mathbf{x}_{k+1}), \boldsymbol{\psi}(\mathbf{u}_k)]$, the

Koopman model becomes $\boldsymbol{\beta}_k = \mathbf{K}\boldsymbol{\alpha}_k$ [21, 35]. We can extract $\mathbf{K}_z$ and $\mathbf{K}_g$ from $\mathbf{K}$ and get

$$\boldsymbol{\phi}\left(\mathbf{x}_{k+1}\right) = \mathbf{K}_z\boldsymbol{\phi}\left(\mathbf{x}_k\right) + \mathbf{K}_g\boldsymbol{\psi}\left(\mathbf{u}_k\right). \tag{1}$$

2.2 Extended Dynamic Mode Decomposition

We need to estimate the Koopman operator using data in a finite-dimensional observation space in order to use it. First, define two data matrices $\mathbf{Y} = \left[\boldsymbol{\alpha}_0 \ \boldsymbol{\alpha}_1 \ \cdots\right] \in \mathbb{R}^{(n_z+n_g)\times N}$ and $\bar{\mathbf{Y}}\left[\boldsymbol{\beta}_0 \ \boldsymbol{\beta}_1 \ \cdots\right] \in \mathbb{R}^{(n_z+n_g)\times N}$, where N is the number of data. $\mathbf{Y}$ contains the "current" data, and $\bar{\mathbf{Y}}$ contains the data one time step further. We can estimate the Koopman matrix $\mathbf{K}$ by solving

$$\min_{\mathbf{K}} ||\mathbf{K}\mathbf{Y} - \bar{\mathbf{Y}}||_F, \tag{2}$$

where F indicates the Frobenius norm. To solve this, EDMD does

$$\mathbf{K} = \bar{\mathbf{Y}}\mathbf{Y}^\top\left(\mathbf{Y}^\top\right)^\dagger \mathbf{Y}^\dagger = \left(\bar{\mathbf{Y}}\mathbf{Y}^\top\right)\left(\mathbf{Y}\mathbf{Y}^\top\right)^\dagger. \tag{3}$$

The matrix $\mathbf{Y}\mathbf{Y}^\top$ is a square matrix and is well-conditioned to matrix inversion computation.

2.3 Recursive Least Squares

Suppose we have a standard linear regression model of the form $y_k = \mathbf{x}_k^\top\mathbf{w} + e_k$, where $y \in \mathbb{R}$ is the scalar output, $\mathbf{x} \in \mathbb{R}^n$ is the input/regressor vector, $\mathbf{w} \in \mathbb{R}^n$ is the unknown parameter vector we wish to estimate, $e \in \mathbb{R}$ is noise or modeling error. To estimate $\mathbf{w}$ in a way that incorporates each new data pair $(\mathbf{x}_k, y_k)$, RLS algorithm solves $\min_{\mathbf{w}} J = \sum_{k=1}^N \left(y_k - \mathbf{x}_k^\top\mathbf{w}\right)$. First, initialize the RLS algorithm at $k = 0$ by picking $\mathbf{w}_0$ and $\mathbf{P}_0$. At each time step k, given the new data pair $(\mathbf{x}_k, y_k)$, update the gain vector $\mathbf{g}_k$, the parameter estimate $\mathbf{w}_k$, and the inverse covariance $\mathbf{P}_k$ by

$$\mathbf{g}_k = \frac{\mathbf{P}_{k-1}\mathbf{x}_k}{1 + \mathbf{x}_k^\top\mathbf{P}_{k-1}\mathbf{x}_k}, \mathbf{w}_k = \mathbf{w}_{k-1} + \mathbf{g}_k\left(y_k - \mathbf{x}_k^\top\mathbf{w}_{k-1}\right), \mathbf{P}_k = \mathbf{P}_{k-1} - \frac{\mathbf{P}_{k-1}\mathbf{x}_k\mathbf{x}_k^\top\mathbf{P}_{k-1}}{1 + \mathbf{x}_k^\top\mathbf{P}_{k-1}\mathbf{x}_k}. \tag{4}$$

Conceptually, $\mathbf{P}_k$ tracks the inverse of the "data covariance" seen so far. Updating $\mathbf{P}$ uses the Sherman-Morrison formula [36] so that large matrix inversions are unnecesary. In Sec. 3.3 and Appx. B, we show that RLS can be generalized to update state-space models.

2.4 Model Predictive Control

A classical MPC policy computes control actions by solving a trajectory optimization problem at each step. It minimizes an objective function composed of a terminal cost and either an integral (for continuous-time) or a summation (for discrete-time) of the running cost, subject to initial conditions, system dynamics, and equality and inequality constraints. Only the first control input from the solution is applied before the process repeats at the next time step. More details are provided in Appx. A.

3 Recursive Koopman Learning

In this section, we present the details of RKL, which is built on an MPC framework. Initially, a small dataset is collected offline and used to compute an initial model via EDMD, which serves to initialize both the MPC and RLS modules. Subsequently, the MPC controller, using the Koopman model, iteratively computes the optimal control input at each time step according to the user-defined control objective. Simultaneously, the RLS module continuously updates the Koopman model in real time using newly collected data. The overall pipeline is shown in Fig. 1 and Alg. 1.

Algorithm 1 Recursive Koopman learning

Require: Initial dataset $\mathbf{Y}_0$ and $\mathbf{Y}_1$
   $\mathbf{K}_0, \mathbf{P}_0 \leftarrow \text{EDMD}\left(\mathbf{Y}_0, \mathbf{Y}_1\right)$
   $\mathbf{u}_0 = \text{MPC}\left(\mathbf{K}_0, \mathbf{z}_0, \bar{\mathbf{z}}\right)$ ▷ $\bar{\mathbf{z}}$ is the goal
   $\mathbf{z}_1 = \text{ENV}\left(\mathbf{z}_0, \mathbf{u}_0\right)$
   $k = 0$
   while True **do**
      $\mathbf{K}_{k+1}, \mathbf{P}_{k+1} \leftarrow \text{RLS}\left(\mathbf{K}_k, \mathbf{P}_k, \mathbf{z}_k, \mathbf{u}_k, \mathbf{z}_{k+1}\right)$
      $\mathbf{u}_{k+1} \leftarrow \text{MPC}\left(\mathbf{K}_{k+1}, \mathbf{z}_{k+1}, \bar{\mathbf{z}}\right)$
      $\mathbf{z}_{k+2} = \text{ENV}\left(\mathbf{z}_{k+1}, \mathbf{u}_{k+1}\right)$
      $k \leftarrow k + 1$
   end while

3.1 MPC Solver

For controller synthesis, we formulate an MPC problem using the learned Koopman model. Any MPC technique could be used; we use Sequential Action Control (MPC-SAC) [37] as the solver because it achieves better performance than LQR in experiments (details in Sec. 4), similar to the findings in [38]. Further details can be found in [37] and Appx. A.

3.2 Dataset Initialization

Previous studies have initialized Koopman models using normal distributions [21], but this often causes numerical issues, such as severely ill-conditioned optimal control problems. In MPC-SAC, such an initialization can cause the adjoint variable to become excessively large, leading to overflow. To avoid this, we initialize the Koopman model using a dataset, ensuring numerical stability during MPC optimization. Sec. 4 details the initial dataset collection and its impact on control performance.

3.3 Real-Time Recursive Model Update

The learned Koopman model is continuously updated while pursuing the control goal. To efficiently utilize runtime data, we aim to find a stable, fast, and lightweight real-time model updating algorithm. RLS (introduced in Sec. 2.3) is well-suited for this purpose: it is tailored for linear systems, much faster and lighter than resolving full linear regressions, and can be readily generalized to update linear state-space models. Appx. Alg. 3 summarizes the algorithm.

Consider the two data matrices used in Eq. 2. At time step k, we have the data matrices $\mathbf{Y}_k$ and $\bar{\mathbf{Y}}_k$, as well as the latest estimate of the Koopman matrix $\mathbf{K}_k$. Additionally, we define

$$\mathbf{Q}_k := \bar{\mathbf{Y}}_k \mathbf{Y}_k^\top, \mathbf{P}_k := \left(\mathbf{Y}_k \mathbf{Y}_k^\top\right)^\dagger . \tag{5}$$

After one time step, we obtain a new pair of data $\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k$ and we use them to update the model $\mathbf{K}_k$. Finding that $\mathbf{P}$ tracks the inverse of the data covariance, similar to RLS (Eq. 4), we can utilize the Sherman-Morrison formula [36] and the update rule of $\mathbf{P}$ is given by

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \gamma_k \mathbf{P}_k \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^\top \mathbf{P}_k, \gamma_k := \frac{1}{1 + \boldsymbol{\alpha}_k^\top \mathbf{P}_k \boldsymbol{\alpha}_k} . \tag{6}$$

It can be shown that the estimate of the Koopman matrix can be updated through

$$\mathbf{K}_{k+1} = \mathbf{K}_k + \gamma_k \left(\boldsymbol{\beta}_k - \mathbf{K}_k \boldsymbol{\alpha}_k\right) \boldsymbol{\alpha}_k^\top \mathbf{P}_k . \tag{7}$$

A single RLS update yields *exactly the same result as retraining the entire model with EDMD* on the updated dataset (see Appx. B for the proof). Our algorithmic analysis (Appx. C) shows that RLS is well-suited for online updates under continuous data growth, as *its speed depends only on the dimension of the observables*, not on dataset size. Additionally, among all methods that compute exact solutions to the linear regression in Eq. 2, RLS has the lowest time complexity. In contrast, EDMD's complexity grows with both the observables dimension and dataset size, making it increasingly slower as more data is added, which is detrimental to real-time model updates under continuous data growth.

Based on the conclusion that RLS and EDMD produce identical computational results, we discuss the convergence properties of RKL by analyzing the convergence of EDMD. In particular, we assume that the dataset forms a Markov chain, which aligns well with our data collection process (Sec. 4) for both the initial and online dataset.

We find that the Strong Law of Large Numbers for Markov chains [32] can be used to prove the convergence of EDMD under continuous data growth. However, this theorem requires the dataset to be an *irreducible*, *aperiodic*, and *positively recurrent* Markov chain w.r.t. an invariant measure μ, meaning the chain is *ergodic*. These requirements are overlooked in previous research [15]. In addition, the observation function must be μ-square-integrable, and the $\mathbf{P}$ matrix in Eq. 5 must be of full rank. Together, these conditions provide sufficient guarantees for the convergence of EDMD as the size of the dataset approaches infinity. Details of the analysis is presented in Appx. D. In the next section, we relate the above results to the experimental results and the ACG hypothesis.

4 Experiments

Our experiments are conducted on a simulated tracking tasks with a planar two-link arm, and dynamic balancing with a Soft Stewart Platform (SSP) [39] (Appx. Fig. 4), a nonlinear hybrid system actuated with soft transmissions [40]. Details are shown in Appx. E.

4.1 Benchmarks

We use the following five methods in our experiments:
- **RKL-SAC** and **RKL-LQR**: Both are RKL, but different MPC solvers are chosen (MPC-SAC or LQR).
- **KL-SAC** and **KL-LQR**: Standard Koopman learning without any model updates [41].
- **NN-MPPI**: Model-based RL using Model Predictive Path Integral (MPPI) control via NN models [11].
- **RL-SAC**: Soft Actor-Critic, a prominent model-free RL algorithm [7].
- **REDQ**: Randomized Ensembled Double Q-Learning, a SotA sample-efficient model-free RL algorithm [42].

4.2 Planar Two-Link Arm Experiments

We evaluate all benchmarks on a planar two-link arm simulated in MuJoCo. The goal is to control the joint states to any desired states in the workspace, and the performance is assessed based on the ability to control the tip to track a 500-step reference trajectory shaped like an "8" (details are shown in Appx. E.1 and Fig. 7-10). Each method is tested across 50 trials, summarized in Table 1. RKL achieves the best overall performance while requiring a significantly smaller dataset compared to RL. Specifically, RKL-SAC outperforms the RL baselines using only 3,500 steps of data—3,000 of which are generated by random actions—representing just ∼5% of the minimum data required by RL methods. Although NN-

Method	Dataset size	RMSE mean (cm)	Time lag mean (sec)
RKL-SAC	500 Demo + 500	0.725	0.01
RKL-SAC	3,000 + 500	0.8359	0.0228
RKL-SAC	5,000 + 500	0.8723	0.018
KL-SAC	5,000	0.901	0.02
RKL-SAC	2,000 + 500	0.9016	0.0196
RKL-SAC	1,000 + 500	1.0396	0.0214
KL-SAC	3,000	1.2037	-0.032
RKL-LQR	5,000 + 500	1.3925	0.1392
RKL-LQR	500 Demo + 500	1.4341	0.16
KL-SAC	2,000	1.5113	-0.031
KL-LQR	5,000	1.5379	0.1354
RKL-LQR	3,000 + 500	1.5478	0.1378
RL-SAC	2,000,000	1.6001	0.15
RKL-SAC	500 + 500	1.67	0.0176
RL-SAC	2,500,000	1.6755	0.17
RKL-LQR	2,000 + 500	1.6887	0.1962
KL-LQR	3,000	1.7607	0.13
RL-SAC	1,500,000	1.7823	0.21
REDQ	80,000	1.8107	0.19
REDQ	100,000	1.9265	0.2
REDQ	60,000	2.2511	0.23
KL-LQR	2,000	2.6088	0.1276
RKL-LQR	1,000 + 500	2.6257	0.188
NN-MPPI	15,000	3.1099	-0.2111666667
KL-SAC	1,000	3.6219	-0.1808
RL-SAC	1,000,000	3.6782	0.35
NN-MPPI	10,000	3.8139	-0.2897
KL-LQR	1,000	4.8095	0.2324
RKL-LQR	500 + 500	5.8193	-0.2148
NN-MPPI	5,000	6.3037	-0.0013
KL-SAC	500	7.5709	-0.7148
KL-SAC	500 Demo	8.1981	-0.2
KL-LQR	500 Demo	8.4155	-0.6
KL-LQR	500	8.5075	-1.0408

Table 1: The results of tracking a reference trajectory on the simulated planar two-link arm, sorted by RMSE. Online model updates significantly improves sampling efficiency and control performance. Additionally, MPC-SAC enhances the performance of both RKL and KL compared to using LQR. RKL shows higher sample efficiency and superior control performance than the baselines.

MPPI requires less data than model-free RL methods (though still more than RKL), its control performance is comparatively lower. Furthermore, using MPC-SAC improves the performance of both RKL and KL relative to LQR, supporting our claim in Sec. 3.1. Comparing KL to RKL demonstrates the impact of online recursive model updates—reducing not only the RMSE by up to 77.94% but also the time lag, especially on small datasets.

We evaluate two types of initial datasets for KL and RKL: a 500-step demonstration dataset (500 Demo) and datasets generated by uniformly sampling actions from the action space. When initialized with the 500 Demo dataset, RKL-SAC outperforms all competing methods, including RKL-SAC trained on larger datasets. This advantage reflects the ACG hypothesis and the sufficient conditions for convergence discussed in Sec. 3.3 and Appx. D. In our convergence analysis, the state can be interpreted broadly to include both the system state and the control input, so that the controller can be treated as part of the dynamics. A closed-loop dynamical system with an ideal policy in the ACG hypothesis for a given control objective can be represented as an autonomous dynamical system in this generalized state space—*the exact target system we aim to learn*. A high-quality demonstration induces a Markov chain that is nearly ergodic w.r.t. the target dynamics' invariant measure. As a result, the initial model's invariant measure closely approximates the true one, providing a good start for rapid convergence to the optimal representation through online model updates.

The success of RKL empirically validates the weak ACG hypothesis, and our convergence analysis helps explain this. While pursuing the control objective, real-time online model updates ensure that the model estimate at each time step is globally optimal, in the sense of Eq. 2, given the currently available data. This also implies that the system is approaching the behavior of a perfect system in a manner we define as optimal at the moment—that is, it is doing its best to converge toward the ergodicity condition (though this is never attainable). We infer that metrics related to ergodicity should be developed to guide the data collection process, thereby ensuring it more closely aligns with the ergodicity requirements. This, in turn, would lead to faster convergence and more sample-efficient learning, particularly for tasks where collecting high-quality demonstration data is challenging.

4.3 Soft Stewart Platform Experiments

Due to the nature of the soft actuators and the mechanical design of the system, the SSP is a highly nonlinear, hybrid, and time-varying dynamical system, making it an ideal platform for testing RKL. A puck—comprised of a solid 38-mm Delrin ball enclosed within a 5-cm-diameter plastic ring—is placed on the SSP. The objective is to control the puck's state, enabling it to balance at a desired location or follow a reference trajectory. Details is presented in Appx. E.2.

The initial dataset is collected by a human operator using a SpaceMouse Compact. The same initial dataset is used for all experiments but cropped to different time lengths (Appx. F). In RKL, each RLS update is completed in 20 ms (limited by the feedback thread).

4.3.1 Puck Balancing

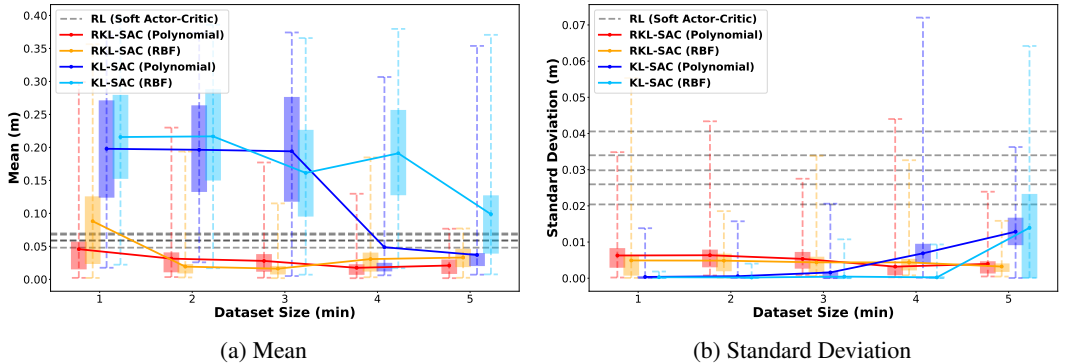|	
(a) Mean	(b) Standard Deviation

Figure 2: KL-SAC vs. RKL-SAC vs. RL-SAC (five random seeds) in puck balancing. The whiskers show the range of the data, the box represents the middle 50%, and the lines connect the averages.

We test the learned controllers at 293 uniformly distributed goal points across the platform surface. Each trial lasts 20 seconds, and the model in RKL is reset at the end of each trial. The mean and standard deviation of the last-5-second absolute position error are used to evaluate the performance at each desired position. Large mean values indicate significant steady-state errors, while large standard deviations reflect substantial oscillations. Two types of observation functions are tested: a 28-dimensional polynomial basis function up to the third degree and a 117-dimensional Gaussian radial basis function (RBF) (Appx. E.2.3). Fig. 3 illustrates a successful example of RKL-SAC.

We compare KL-SAC, RKL-SAC, and RL-SAC across varying dataset sizes. As shown in Fig. 2a, KL-SAC struggles with datasets under 4 minutes, while RKL-SAC remains robust except for using RBF with 1-minute data. Fig. 2b shows KL-SAC and RKL-SAC have low standard deviations, indicating stable puck behavior, whereas RL-SAC results in larger oscillations. However, KL-SAC's low deviation with small datasets reflects puck immobilization due to static friction, not true stability—online updates help overcome this. Notably, RKL-SAC achieves better performance than RL-SAC using just 1 minute of initial data and 20 seconds of online updates (8,000 steps), compared to RL-SAC's 2 hours 46 minutes (100,000 steps)—only 0.8% of the training time. We also test NN-MPPI

under the same setup [43]. RKL improves upon NN-MPPI with a 66% lower average error and a 8 times better standard deviation, using only 2.6% of the training time.

We also evaluate RKL-SAC on never-before-seen states by removing all data with positions in the first quadrant from a 4-minute dataset (Appx. Fig. 5). Using the polynomial observation function, RKL-SAC achieves a mean error of 0.03 m and a standard deviation of 0.0237 m across 293 test points. Online model updates quickly estimate the Koopman model in regions that is not presented in the dataset. In contrast, KL-SAC performs poorly with the same cropped dataset. Although RKL-SAC does not outperform than using the full dataset, it still exceeds RL-SAC, demonstrating strong stability and sample efficiency.
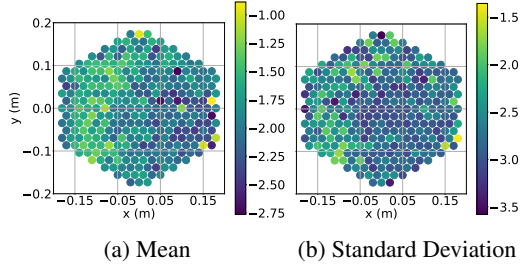

(a) Mean   (b) Standard Deviation

Figure 3: The mean and standard deviation of the absolute errors (scaled by log base 10) computed over the last 5 seconds of each 20-second RKL-SAC trial. This controller uses a 4-minute-long initial dataset and a 28-dimensional polynomial observation function. The distance between each goal point and its six adjacent goal points is 2 cm.

Again, online model updates are critical to RKL's strong performance. A KL-SAC controller trained on a larger initial dataset does not outperform an RKL-SAC controller trained on a smaller initial dataset with 20-second model updates, which means 20 seconds of online data collected during the process of achieving the control goal is more effective than several minutes of manually collected data, as shown in Fig. 2a. This empirically validates the weak ACG hypothesis. Real-time model updates can efficiently utilize this data and enhance the controller's performance.

4.3.2 Tracking of Trajectories Containing Contacts

We extend our study to control the puck such that it follows a reference trajectory—one is shaped like the letter "N" and involves contacts with the boundaries (Appx. Fig. 6). To encourage contacts, the start, end, and both corners of the "N" lie outside the platform's boundaries. The reference velocity for each segment of the "N" is constant, with each segment taking 7 seconds to complete.

Method	Mean (m)	Max (m)	Min (m)
KL-SAC (4 min)	0.1079	0.2319	0.0628
RKL-SAC (4 min)	**0.0636**	**0.0812**	**0.0477**
RL-SAC	0.0821	0.1006	0.0732

Table 2: KL-SAC vs. RKL-SAC vs. RL-SAC in tracking an "N" trajectory.

We conduct ten trials per method, starting from the lower-left corner of the "N". For each method, we select the five best trials based on the Fréchet distance between the reference and puck's position trajectories, and report the mean, maximum, and minimum value (Table 2). The RKL-SAC and KL-SAC controllers use the a 4-minute dataset, while the RL-SAC policy is chosen based on the lowest mean error. RKL-SAC achieves the best performance. Trajectories from the best trials are shown in Appx. Fig. 6.

5   Conclusion

In this paper, we present a sample-efficient Koopman-based learning pipeline, RKL. This pipeline can rapidly generate a controller capable of not only controlling hybrid nonlinear dynamical systems but also quickly learning and recursively updating models in real time. Our experiments on a simulated planar two-link arm and a soft-actuated hardware platform demonstrate the superior sample efficiency and improved control performance of RKL compared to SotA benchmarks. Additionally, we present the first formal convergence analysis of EDMD and RLS under continuous data growth within Markov chains, providing detailed sufficient conditions for convergence—critical for practical use and absent in prior work. We not only empirically validate the ACG hypothesis but also use our analysis results to explain it. Furthermore, we provide detailed algorithmic analyses of EDMD and RLS to prove that RKL is lightweight and fast, with complexity independent of dataset size.

6 Limitations

While model updating can significantly improve the control performance, we have observed cases where the RKL controller struggles to move the puck away from its current location during experiments on the SSP. This typically occurs when the puck remains in a small region for an extended period. As a result, the dataset becomes imbalanced: the model achieves high accuracy in predicting state transitions near the current location and at zero velocity but performs worse elsewhere. Additionally, because it is difficult to collect demonstration data on the SSP—whether from human operators or teaching policies—the initial model often ends up with an invariant measure that is far from the true distribution, as discussed in Sec. 4.2. Therefore, such an initial model is a poor starting point for RKL and can lead to the mentioned problems above.

Based on our convergence analysis, this issue can potentially be addressed by using high-quality demonstration data or by applying ergodicity-related metrics to guide the controller's behavior while attempting the control goal. Many systems are naturally poorly suited for learning from demonstration, making ergodicity-related metrics especially important in those cases.

Acknowledgments

References

[1] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti. High-frequency nonlinear model predictive control of a manipulator. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7330–7336, 2021. doi:10.1109/ICRA48506.2021.9560990.

[2] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter. Perceptive locomotion through nonlinear model-predictive control. *IEEE Transactions on Robotics*, 39(5):3402–3421, 2023. doi:10.1109/TRO.2023.3275384.

[3] A. Meduri, P. Shah, J. Viereck, M. Khadiv, I. Havoutis, and L. Righetti. BiConMP: A nonlinear model predictive control framework for whole body motion planning. *IEEE Transactions on Robotics*, 39(2):905–922, 2023. doi:10.1109/TRO.2022.3228390.

[4] S. Le Cleac'h, T. A. Howell, S. Yang, C.-Y. Lee, J. Zhang, A. Bishop, M. Schwager, and Z. Manchester. Fast contact-implicit model predictive control. *IEEE Transactions on Robotics*, 40:1617–1629, 2024. doi:10.1109/TRO.2024.3351554.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[6] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

[8] R. Jitosho, T. G. W. Lum, A. Okamura, and K. Liu. Reinforcement learning enables real-time planning and control of agile maneuvers for soft robot arms. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1131–1153. PMLR, 06–09 Nov 2023.

[9] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi:10.15607/RSS.2018.XIV.049.

[10] Y. Chen, T. Wu, S. Wang, X. Feng, J. Jiang, Z. Lu, S. McAleer, H. Dong, S.-C. Zhu, and Y. Yang. Towards human-level bimanual dexterous manipulation with reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 5150–5163. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/217a2a387f52c30755c37b0a73430291-Paper-Datasets_and_Benchmarks.pdf.

[11] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017. doi:10.1109/ICRA.2017.7989202.

[12] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020. doi:10.1126/scirobotics.abc5986.

[13] J. W. Mock and S. S. Muknahallipatna. A comparison of PPO, TD3 and SAC reinforcement algorithms for quadruped walking gait generation. *Journal of Intelligent Learning Systems and Applications*, 15(1):36–56, 2023. doi:10.4236/jilsa.2023.151003.

[14] J. H. Tu. *Dynamic mode decomposition: Theory and applications*. PhD thesis, Princeton University, 2013. URL http://turing.library.northwestern.edu/login?url=https://www.proquest.com/dissertations-theses/dynamic-mode-decomposition-theory-applications/docview/1458341928/se-2.

[15] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25:1307–1346, 2015. doi:10.1007/s00332-015-9258-5.

[16] C. W. Rowley and S. T. Dawson. Model reduction for flow analysis and control. *Annual Review of Fluid Mechanics*, 49(49):387–417, 2017. ISSN 1545-4479. doi:10.1146/annurev-fluid-010816-060042.

[17] N. Geneva and N. Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022. ISSN 0893-6080. doi:10.1016/j.neunet.2021.11.022.

[18] Y. Susuki and I. Mezić. Nonlinear Koopman modes of coupled swing dynamics and coherency identification. In *IEEE PES General Meeting*, pages 1–8, 2010. doi:10.1109/PES.2010.5589363.

[19] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan. Modeling and control of soft robots using the Koopman operator and model predictive control. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019. doi:10.15607/RSS.2019.XV.060.

[20] D. Bruder, D. Bombara, and R. J. Wood. A Koopman-based residual modeling approach for the control of a soft robot arm. *The International Journal of Robotics Research*, 0(0), 2024. doi:10.1177/02783649241272114.

[21] I. Abraham and T. D. Murphey. Active learning of dynamics for data-driven control using Koopman operators. *IEEE Transactions on Robotics*, 35(5):1071–1083, 2019. doi:10.1109/TRO.2019.2923880.

[22] G. Mamakoukas, M. L. Castaño, X. Tan, and T. D. Murphey. Derivative-based Koopman operators for real-time control of robotic systems. *IEEE Transactions on Robotics*, 37(6):2173–2192, 2021. doi:10.1109/TRO.2021.3076581.

[23] F. Li, A. Abuduweili, Y. Sun, R. Chen, W. Zhao, and C. Liu. Continual learning and lifting of Koopman dynamics for linear control of legged robots. In *Proceedings of the 7th Annual Learning for Dynamics and Control Conference*, volume 283 of *Proceedings of Machine Learning Research*, pages 1–30. PMLR, 4–6 June 2025.

[24] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

[25] M. H. Hayes. Chapter 9.4 - recursive least squares. In *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 1996. doi:10.1080/00401706.1997.10485128.

[26] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. In *2020 Conference on Robot Learning (CoRL)*, pages 1101–1112. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/nagabandi20a.html.

[27] H. Zhang, C. W. Rowley, E. A. Deem, and L. N. Cattafesta. Online dynamic mode decomposition for time-varying systems. *SIAM Journal on Applied Dynamical Systems*, 18(3):1586–1609, 2019. doi:10.1137/18M1192329.

[28] H. M. Calderón, E. Schulz, T. Oehlschlägel, and H. Werner. Koopman operator-based model predictive control with recursive online update. In *2021 European Control Conference (ECC)*, pages 1543–1549, 2021. doi:10.23919/ECC54610.2021.9655220.

[29] F. Nüske, S. Peitz, F. Philipp, M. Schaller, and K. Worthmann. Finite-data error bounds for Koopman-based prediction and control. *Journal of Nonlinear Science*, 33(1):14, 2023. doi:10.1007/s00332-022-09862-1.

[30] C. Zhang and E. Zuazua. A quantitative analysis of Koopman operator methods for system identification and predictions. *Comptes Rendus. Mécanique*, 351(S1):1–31, 2023. doi:10.5802/crmeca.138.

[31] F. M. Philipp, M. Schaller, S. Boshoff, S. Peitz, F. Nüske, and K. Worthmann. Extended dynamic mode decomposition: Sharp bounds on the sample efficiency. *arXiv preprint arXiv:2402.02494*, 2024. URL https://arxiv.org/abs/2402.02494.

[32] L. Breiman. The strong law of large numbers for a class of Markov chains. *The Annals of Mathematical Statistics*, 31(3):801–803, 1960. URL https://www.jstor.org/stable/2237593.

[33] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931. doi:10.1073/pnas.17.5.315.

[34] H. H. Asada. Global, unified representation of heterogenous robot dynamics using composition operators: A Koopman direct encoding method. *IEEE/ASME Transactions on Mechatronics*, 28(5):2633–2644, 2023. doi:10.1109/TMECH.2023.3253599.

[35] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016. doi:10.1137/15M1013857.

[36] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950. ISSN 00034851. URL http://www.jstor.org/stable/2236561.

[37] A. R. Ansari and T. D. Murphey. Sequential action control: Closed-form optimal control for nonlinear and nonsmooth systems. *IEEE Transactions on Robotics*, 32(5):1196–1214, 2016. doi:10.1109/TRO.2016.2596768.

[38] H. Nishimura and M. Schwager. SACBP: Belief space planning for continuous-time dynamical systems via stochastic sequential action control. *The International Journal of Robotics Research*, 40(10-11):1167–1195, 2021. doi:10.1177/02783649211037697.

[39] J. Ketchum, J. Avtges, M. Schlafly, H. Young, T. Kim, R. L. Truby, and T. D. Murphey. Force and speed in a soft stewart platform, 2025. URL https://arxiv.org/abs/2504.13127.

[40] T. Kim, P. Kaarthik, and R. L. Truby. A flexible, architected soft robotic actuator for motorized extensional motion. *Advanced Intelligent Systems*, 6(11):2300866, 2024. doi:10.1002/aisy.202300866.

[41] I. Abraham, G. de la Torre, and T. Murphey. Model-based control using Koopman operators. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017. doi: 10.15607/RSS.2017.XIII.052.

[42] X. Chen, C. Wang, Z. Zhou, and K. W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=AY8zfZm0tDd.

[43] J. Avtges, J. Ketchum, M. Schlafly, H. Young, T. Kim, A. Pinosky, R. L. Truby, and T. D. Murphey. Real-time reinforcement learning for dynamic tasks with a parallel soft robot. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

[44] S. Boyd and L. Vandenberghe. Part III least squares. In *Introduction to applied linear algebra: vectors, matrices, and least squares*. Cambridge university press, 2018. URL https://web.stanford.edu/~boyd/vmls/vmls.pdf.

[45] G. W. Stewart. On the continuity of the generalized inverse. *SIAM Journal on Applied Mathematics*, 17(1):33–45, 1969. ISSN 00361399. URL http://www.jstor.org/stable/2099241.

[46] F. Foundation. Reacher - gymnasium documentation. https://gymnasium.farama.org/environments/mujoco/reacher/.

[47] T. A. Berrueta, A. Pinosky, and T. D. Murphey. Maximum diffusion reinforcement learning. *Nature Machine Intelligence*, 6(5):504–514, 2024.

[48] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[49] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016. doi:10.1109/ICRA.2016.7487277.

A Sequential Action Control

A continuous-time MPC policy determines control inputs for a dynamical system by repeatedly solving a constrained trajectory optimization problem. At each iteration, we solve

$$\min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)} \quad J = \underbrace{m\left(\mathbf{x}\left(T\right)\right)}_{\text{terminal cost}} + \int_0^T \underbrace{l\left(\mathbf{x}\left(\tau\right),\mathbf{u}\left(\tau\right)\right)}_{\text{running cost}} d\tau$$

$$\text{s.t.} \quad \mathbf{x}\left(0\right) = \mathbf{x}_{\text{init}}, \text{ initial condition,}$$
$$\dot{\mathbf{x}}\left(t\right) = \mathbf{f}(\mathbf{x}\left(t\right),\mathbf{u}\left(t\right)), \text{ dynamics,}$$
$$\mathbf{c}_{\text{eq}}(\mathbf{x}\left(t\right),\mathbf{u}\left(t\right)) \leq \mathbf{0}, \text{ equality constraints,}$$
$$\mathbf{c}_{\text{iq}}(\mathbf{x}\left(t\right),\mathbf{u}\left(t\right)) = \mathbf{0}, \text{ inequality constraints.}$$

(8)

Once the optimal control sequence is obtained, only the first control input is applied to the system, and the process is repeated at the next time step.

Sequential Action Control (MPC-SAC) [37] is a model-based approach that solves MPC in real time within a closed-loop framework for nonlinear systems. Instead of iteratively optimizing finite-horizon control sequences to minimize a given objective, MPC-SAC utilizes a closed-form expression to determine individual control actions. These actions, applied over short durations, are designed to optimally enhance a tracking objective over a long time horizon. Moreover, the existence and uniqueness of optimal actions are globally guaranteed. Here, we introduce the algorithm in

the context of Koopman-based control, and the detailed derivation is presented in [37, 21]. The algorithm is summarized in Alg. 2.

Algorithm 2 MPC-SAC using a Koopman model

Require: Nominal policy $\boldsymbol{\mu}$, weight matrix $\bar{\mathbf{R}}$, Koopman matrix $\mathbf{K}$, current feedback $\mathbf{z}$, reference $\mathbf{z}_{\text{ref}}$ and $\mathbf{u}_{\text{ref}}$, time step length δt, horizon length H
  $\mathbf{u}^* = \text{MPC-SAC}\left(\mathbf{K}, \mathbf{z}, \mathbf{z}_{\text{ref}}, \mathbf{u}_{\text{ref}}, \boldsymbol{\mu}, \bar{\mathbf{R}}, \delta t, H\right)$
    $\boldsymbol{\mu}$.update $\left(\mathbf{K}, \mathbf{z}_{\text{ref}}, \mathbf{u}_{\text{ref}}\right)$
    Adjoint.reset $\left(\delta t, H\right)$
    Objective.reset $\left(\mathbf{z}_{\text{ref}}, \mathbf{u}_{\text{ref}}\right)$
    traj $= \boldsymbol{\mu}$.simulate $\left(H, \delta t, \text{Objective}\right)$
    $\boldsymbol{\rho}$_traj=Adjoint.simulate $\left(\text{traj}\right)$
    $\mathbf{u} = \boldsymbol{\mu}$.solve $\left(\mathbf{z}, \text{Objective}\right)$
    $\mathbf{u}^* = -\bar{\mathbf{R}}^{-1}\mathbf{K}_u^{\top}\boldsymbol{\rho}\left(0\right) + \mathbf{u}$

The dynamics model used to synthesize the controller is a continuous-time Koopman model,

$$\dot{\mathbf{z}}\left(t\right) = \mathbf{f}\left(\mathbf{z}\left(t\right), \mathbf{u}\left(t\right)\right) = \mathbf{K}_z\mathbf{z}\left(t\right) + \mathbf{K}_u\mathbf{u}\left(t\right),\tag{9}$$

where $\mathbf{z}$ is the observation of the state. Define the objective function $J_1 \in \mathbb{R}$,

$$J_1 = \int_{t_0}^{t_0+T} l\left(\mathbf{z}\left(s\right), \boldsymbol{\mu}\left(\mathbf{z}\left(s\right)\right)\right) ds + m\left(\mathbf{z}\left(t_0 + T\right)\right),\tag{10}$$

where $\boldsymbol{\mu}\left(\mathbf{z}\right)$ is a nominal feedback control policy. We use LQR as the nominal policy in our experiments.

MPC-SAC finds the change in the objective (Eq. 10) due to inserting a new control policy $\boldsymbol{\mu}^*$ into the nominal trajectory for a short duration around $t = \tau \in [t_0, t_0 + T]$. The related mode insertion gradient is given by

$$\frac{\partial J_1}{\partial \lambda}\big|_{t=\tau,\lambda=0} = \boldsymbol{\rho}\left(\tau\right)^{\top}\left(\mathbf{f}_2 - \mathbf{f}_1\right),\tag{11}$$

where $\mathbf{f}_1 = \mathbf{f}\left(\mathbf{z}\left(\tau\right), \boldsymbol{\mu}\left(\mathbf{z}\left(\tau\right)\right)\right)$ and $\mathbf{f}_2 = \mathbf{f}\left(\mathbf{z}\left(\tau\right), \boldsymbol{\mu}^*\left(\mathbf{z}\left(\tau\right)\right)\right)$. The adjoint variable (or costate) $\boldsymbol{\rho}$ satisfies

$$\dot{\boldsymbol{\rho}} = -\left(\frac{\partial l}{\partial \mathbf{z}} + \frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}}^{\top}\frac{\partial l}{\partial \mathbf{u}}\right) - \left(\frac{\partial \mathbf{f}}{\partial \mathbf{z}} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}}\right)^{\top}\boldsymbol{\rho},\tag{12}$$

with the terminal condition

$$\boldsymbol{\rho}\left(t_0 + T\right) = \frac{\partial}{\partial \mathbf{z}}m\left(\mathbf{z}\left(t_0 + T\right)\right).\tag{13}$$

We first use the nominal policy to simulate the system's trajectory over the horizon length, starting from the current state. Then, we compute the trajectory of $\boldsymbol{\rho}$ backwards from the terminal condition.

Finally, MPC-SAC solves a new optimization problem to minimize the mode insertion gradient,

$$J_2 = \int_{t_0}^{t_0+T} \frac{\partial J_1}{\partial \lambda}\big|_{t=s,\lambda=0} + \frac{1}{2}\left\|\boldsymbol{\mu}^*\left(s\right) - \boldsymbol{\mu}\left(\mathbf{z}\left(t\right)\right)\right\|_{\bar{R}}^2 ds,\tag{14}$$

where $\bar{\mathbf{R}}$ is a diagonal matrix to bound the change from $\boldsymbol{\mu}$ to $\boldsymbol{\mu}^*$. It can be shown that the analytical solution is [37, 21]

$$\boldsymbol{\mu}^*\left(t\right) = -\bar{\mathbf{R}}^{-1}\mathbf{K}_u^{\top}\boldsymbol{\rho}\left(t\right) + \boldsymbol{\mu}\left(\mathbf{z}\left(t\right)\right).\tag{15}$$

Finally, $\boldsymbol{\mu}^*\left(0\right)$ is applied to the system.

B RLS Update is Equivalent to EDMD

Algorithm 3 Online Model Update using RLS

$\mathbf{K}_{\text{new}}, \mathbf{P}_{\text{new}} = \text{RLS}(\mathbf{K}, \mathbf{P}, \mathbf{z}, \mathbf{u}, \bar{\mathbf{z}})$

$\quad \boldsymbol{\alpha} = [\mathbf{z}; \mathbf{u}]$ ▷ column vector

$\quad \boldsymbol{\beta} = [\bar{\mathbf{z}}; \mathbf{u}]$ ▷ column vector

$\quad \gamma = 1/\left(1 + \boldsymbol{\alpha}^{\top} \mathbf{P} \boldsymbol{\alpha}\right)$

$\quad \mathbf{P}_{\text{new}} = \mathbf{P} - \gamma \mathbf{P} \boldsymbol{\alpha} \boldsymbol{\alpha}^{\top} \mathbf{P}$

$\quad \mathbf{K}_{\text{new}} = \mathbf{K} + \gamma \left(\boldsymbol{\beta} - \mathbf{K} \boldsymbol{\alpha}\right) \boldsymbol{\alpha}^{\top} \mathbf{P}$

Here we prove that the RLS update algorithm for Koopman models (Alg. 3) is mathematically equivalent to retraining a Koopman model using EDMD. Similar proofs are also presented by [27, 28].

Consider the two data matrices used in EDMD. At time step k, assume that we have data matrices $\mathbf{Y}_k$ and $\bar{\mathbf{Y}}_k$, and the latest estimate of the Koopman matrix $\mathbf{K}_k$. In addition, we define

$$\mathbf{Q}_k := \bar{\mathbf{Y}}_k \mathbf{Y}_k^{\top}, \mathbf{P}_k := \left(\mathbf{Y}_k \mathbf{Y}_k^{\top}\right)^{\dagger}. \tag{16}$$

Assume that we want to update the model after one time step. We got a new pair of data $\{\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k\}$, and the data matrices becomes

$$\mathbf{Y}_{k+1} = \begin{bmatrix} \mathbf{Y}_k & \boldsymbol{\alpha}_k \end{bmatrix}, \tag{17}$$

$$\bar{\mathbf{Y}}_{k+1} = \begin{bmatrix} \bar{\mathbf{Y}}_k & \boldsymbol{\beta}_k \end{bmatrix}. \tag{18}$$

Now we can calculate the new $\mathbf{Q}$ and $\mathbf{P}^{\dagger}$

$$\mathbf{Q}_{k+1} = \begin{bmatrix} \bar{\mathbf{Y}}_k & \boldsymbol{\beta}_k \end{bmatrix} \begin{bmatrix} \mathbf{Y}_k^{\top} \\ \boldsymbol{\alpha}_k^{\top} \end{bmatrix} = \bar{\mathbf{Y}}_k \mathbf{Y}_k^{\top} + \boldsymbol{\beta}_k \boldsymbol{\alpha}_k^{\top}, \tag{19}$$

$$\mathbf{P}_{k+1}^{\dagger} = \begin{bmatrix} \mathbf{Y}_k & \boldsymbol{\alpha}_k \end{bmatrix} \begin{bmatrix} \mathbf{Y}_k^{\top} \\ \boldsymbol{\alpha}_k^{\top} \end{bmatrix} = \mathbf{Y}_k \mathbf{Y}_k^{\top} + \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^{\top}. \tag{20}$$

Now we want to update the model. Apply the EDMD update, we get

$$\mathbf{K}_{k+1} = \mathbf{Q}_{k+1} \mathbf{P}_{k+1}. \tag{21}$$

However, computing this needs computing the inverse of a large matrix, and involves several large matrix multiplications. To avoid this, first recall the Sherman-Morrison formula [36]. Suppose $\mathbf{A}$ is an invertible square real matrix, and $\mathbf{u}$, $\mathbf{v}$ are column real vectors. Then $\mathbf{A} + \mathbf{u}\mathbf{v}^{\top}$ is invertible if and only if $1 + \mathbf{v}^{\top} \mathbf{A}^{-1} \mathbf{u} \neq 0$. In this case,

$$\left(\mathbf{A} + \mathbf{u}\mathbf{v}^{\top}\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{\top}\mathbf{A}^{-1}}{1 + \mathbf{v}^{\top}\mathbf{A}^{-1}\mathbf{u}}. \tag{22}$$

This formula can be used to compute $\mathbf{P}_{k+1}$. Start from Eq. 20,

$$\mathbf{P}_{k+1} = \left(\mathbf{P}_k^{\dagger} + \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^{\top}\right)^{-1}$$

$$= \mathbf{P}_k - \frac{\mathbf{P}_k \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^{\top} \mathbf{P}_k}{1 + \boldsymbol{\alpha}_k^{\top} \mathbf{P}_k \boldsymbol{\alpha}_k} \tag{23}$$

$$= \mathbf{P}_k - \gamma_k \mathbf{P}_k \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^{\top} \mathbf{P}_k,$$

$$\gamma_k := \frac{1}{1 + \boldsymbol{\alpha}_k^{\top} \mathbf{P}_k \boldsymbol{\alpha}_k}. \tag{24}$$

By substituting Eq. 19 and 23 into Eq. 21 and simplifing the expression, we can update $\mathbf{K}_k$ without calculating any matrix inverses,

$$\mathbf{K}_{k+1} = \mathbf{K}_k + \gamma_k \left(\boldsymbol{\beta}_k - \mathbf{K}_k \boldsymbol{\alpha}_k\right) \boldsymbol{\alpha}_k^{\top} \mathbf{P}_k. \tag{25}$$

Note that we require $\mathbf{P}$ to be always full rank.

C Algorithmic Complexity Analysis

C.1 EDMD Complexity

After collecting $N + 1$ data pairs $\{\mathbf{x}, \mathbf{u}\}$, let $\mathbf{Y}$ and $\bar{\mathbf{Y}}$ be the matrices of observables (same as Eq. 16), each of size $n \times N$, where n is the dimension of the observation function. The standard EDMD estimate involves:

- $\bar{\mathbf{Y}} \mathbf{Y}^\top$ and $\mathbf{Y} \mathbf{Y}^\top$ each require multiplying an $n \times N$ matrix by an $N \times n$ matrix, which costs $\mathcal{O}\left(n^2 N\right)$.
- Inverting (or pseudo-inverting) $\mathbf{Y} \mathbf{Y}^\top$ to get $\mathbf{P}$ typically costs $\mathcal{O}\left(n^3\right)$.

Hence, updating $\mathbf{K}$ with every new snapshot in a purely retraining fashion would incur $\mathcal{O}\left(n^2 N\right) + \mathcal{O}\left(n^3\right)$ work each time.

C.2 RLS Update Complexity

By contrast, the RLS update uses rank-1 updates and the Sherman–Morrison formula to avoid expensive matrix inversions at every step. At time $k \to k + 1$, the key operations include:

- Updating $\mathbf{P}_{k+1}$ via the Sherman–Morrison formula, which requires a small number of matrix-vector multiplications with matrices and vectors of size n, costing $\mathcal{O}\left(n^2\right)$.
- Updating $\mathbf{K}_{k+1}$. Like the previous step, this is dominated by a small number of matrix-vector multiplications and outer products at $\mathcal{O}\left(n^2\right)$ each update.

Overall, each update has a per-step cost of $\mathcal{O}\left(n^2\right)$.

D Convergence Analysis

D.1 Definitions

$\mathcal{S}$ is the state space. The samples in the dataset construct a Markov chain $\{\mathbf{s}_k\} \subset \mathcal{S}$ satisfying $\mathbf{s}_{k+1} = \mathbf{f}\left(\mathbf{s}_k\right)$ for some map $\mathbf{f} : \mathcal{S} \to \mathcal{S}$. If control inputs are considered, $\mathbf{s}$ contains both the robot state $\mathbf{x}$ and the control input $\mathbf{u}$. In addition, $\boldsymbol{\varphi} : \mathcal{S} \to \mathbb{R}^{n_\varphi}$ is a finite-dimensional observation function applied on $\mathbf{s}$.

The chain $\{\mathbf{s}_k\}$ admits a stationary distribution (invariant measure) μ — if the system starts in the stationary distribution, it remains in that distribution after any number of transitions. Intuitively, we can think of this distribution as describing where the system tends to spend its time. It is an inherent property of the dynamical system and becomes apparent as time approaches infinity. Under the invariant measure μ, define

$$\mathbf{Q} = \mathbb{E}_\mu \left[\boldsymbol{\varphi}\left(\mathbf{f}\left(\mathbf{s}\right)\right) \boldsymbol{\varphi}\left(\mathbf{s}\right)^\top \right], \tag{26}$$

$$\mathbf{P} = \left(\mathbb{E}_\mu \left[\boldsymbol{\varphi}\left(\mathbf{s}\right) \boldsymbol{\varphi}\left(\mathbf{s}\right)^\top \right] \right)^\dagger. \tag{27}$$

The optimal estimate of the Koopman matrix in the mean-square sense in the feature space defined by $\boldsymbol{\varphi}$ is

$$\mathbf{K}^* = \underset{\mathbf{K}}{\operatorname{argmin}} \ \mathbb{E}_\mu \left[\left\| \boldsymbol{\varphi}\left(\mathbf{f}\left(\mathbf{s}\right)\right) - \mathbf{K} \boldsymbol{\varphi}\left(\mathbf{s}\right) \right\|^2 \right]. \tag{28}$$

And it can be calculated by [44]

$$\mathbf{K}^* = \mathbf{Q}\mathbf{P}. \tag{29}$$

To estimate Eq. 26 and 27 from data, based on EDMD and Eq. 16, we use

$$\mathbf{Q}_N = \frac{1}{N} \sum_{k=0}^{N-1} \boldsymbol{\varphi}\left(\mathbf{f}\left(\mathbf{s}_k\right)\right) \boldsymbol{\varphi}\left(\mathbf{s}_k\right)^\top, \tag{30}$$

16

$$\mathbf{P}_N = \left(\frac{1}{N} \sum_{k=0}^{N-1} \boldsymbol{\varphi}\left(\mathbf{s}_k\right) \boldsymbol{\varphi}\left(\mathbf{s}_k\right)^\top \right)^\dagger. \tag{31}$$

where N is the number of data. Then the estimated Koopman matrix calculated by EDMD and RLS is (Appendix B)

$$\mathbf{K}_N = \mathbf{Q}_N \mathbf{P}_N. \tag{32}$$

Our goal is to show $\mathbf{K}_N$ converges to the optimal estimate $\mathbf{K}^*$.

D.2 Assumptions

The following assumptions are required in order to guarantee convergence.
- $\{\mathbf{s}_k\}$ is a Markov chain. It is irreducible, aperiodic, and positive recurrent w.r.t. its invariant measure μ. This implies the chain is ergodic.
- The observation function $\boldsymbol{\varphi}\left(\mathbf{s}\right)$ is μ-square-integrable, i.e.,

$$\mathbb{E}_\mu \left[\|\boldsymbol{\varphi}\left(\mathbf{s}\right)\|^2 \right] = \int_{\mathcal{S}} \|\boldsymbol{\varphi}\left(\mathbf{s}\right)\|^2 d\mu\left(\mathbf{s}\right) < \infty. \tag{33}$$

- $\mathbf{P}$ and $\mathbf{P}_N$ are invertible, i.e. they are both full-rank.

In our experiments, data are collected via continuous sampling at very short time intervals (10 ms). Consequently, the resulting data trajectories can be approximated as Markov chains, consistent with our state-space model. The experimental results indicate the feasibility of this assumption.

Irreducibility requires that from any sampled state $\mathbf{s}$, there is a nonzero probability of reaching any other region of the state space within a finite number of steps. This condition can be satisfied by the two experimental systems we use because both of them are controllable systems over a bounded domain.

A Markov chain is aperiodic if there is no fixed cycle governing the return of states. For controlled dynamical systems, whether this condition can be satisfied is related to the control policy.

A state $\mathbf{s}$ in a Markov chain is positive recurrent if, starting from $\mathbf{s}$, the chain is guaranteed to return to $\mathbf{s}$ at some future time. For fully actuated systems with bounded control inputs and state space, such as the two used in this study, we can reasonably expect the resulting Markov chains to be positive recurrent.

Regarding the assumption about the rank of $\mathbf{P}$ and $\mathbf{P}_N$, since the components of the observation function form an orthonormal set of basis functions and real mechanical systems exhibit uncertainty and noise, this assumption is always satisfied in practice. In our experiments, we check the rank of $\mathbf{P}_N$ and find that it is always full-rank.

D.3 Convergence

Our assumptions enable us to apply the Strong Law of Large Numbers for Markov chains [32] to the sequence $\{\mathbf{s}_k\}$. The SLLN states that for any integrable function $h : \mathcal{S} \to \mathbb{R}$, the following statement holds under our assumptions,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} h\left(\mathbf{s}_k\right) = \mathbb{E}_\mu\left[h\left(\mathbf{s}\right)\right], \text{ almost surely.} \tag{34}$$

Thus for each fixed $i, j \in \{1, ..., n_\varphi\}$,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} \varphi_i\left(\mathbf{s}_k\right) \varphi_j\left(\mathbf{s}_k\right)^\top = \mathbb{E}_\mu\left[\varphi_i\left(\mathbf{s}\right) \varphi_j\left(\mathbf{s}\right)^\top\right], \tag{35}$$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} \varphi_i\left(\mathbf{s}_k\right) \varphi_j\left(\mathbf{s}_{k+1}\right)^\top = \mathbb{E}_\mu\left[\varphi_i\left(\mathbf{s}\right) \varphi_j\left(\mathbf{f}\left(\mathbf{s}\right)\right)^\top\right], \tag{36}$$

almost surely. Hence, we can determine that

$$\mathbf{Q}_N \xrightarrow[N\to\infty]{\text{a.s.}} \mathbf{Q}, \mathbf{P}_N^{-1} \xrightarrow[N\to\infty]{\text{a.s.}} \mathbf{P}^{-1}. \tag{37}$$

We use -1 instead of $\dagger$ here because we have required $\mathbf{P}$ to be full-rank. By standard results on the matrix inverse [45],

$$\mathbf{P}_N \xrightarrow[N\to\infty]{\text{a.s.}} \mathbf{P}. \tag{38}$$

Finally, we can conclude that

$$\mathbf{K}_N = \mathbf{Q}_N\mathbf{P}_N \xrightarrow[N\to\infty]{\text{a.s.}} \mathbf{QP} = \mathbf{K}^*. \tag{39}$$

E   Details of Experiments

E.1   Planar Two-Link Arm

The simulation environment is a customized MuJoCo environment based on the Gymnasium Reacher-v4 environment [46], with modifications to the observations, reward function, action space, and time step length.

Instead of observing the sine and cosine of the joint angles, as in Reacher-v4, our customized environment directly observes the joint angles in radians. To enable joint position control, the new reward function is defined as:

$$r = -100.0\,\|\mathbf{q}_d - \mathbf{q}\| - 10.0\,\|\dot{\mathbf{q}}_d - \dot{\mathbf{q}}\| - u_1^2 - u_2^2, \tag{40}$$

where $\mathbf{q}_d \in \mathbb{R}^2$ and $\dot{\mathbf{q}}_d \in \mathbb{R}^2$ are the desired joint positions and velocities, $\mathbf{q} \in \mathbb{R}^2$ and $\dot{\mathbf{q}} \in \mathbb{R}^2$ are the actual joint positions and velocities, and $u_1 \in \mathbb{R}$ and $u_2 \in \mathbb{R}$ are the joint actuator torques. The action space is defined as $u_1, u_2 \in [-0.5, 0.5]$ Nm. The time step length is set to 10 ms.

During training, the initial positions of the two joints are initialized by uniformly sampling from $[-0.1, 0.1]$ rad, and the initial joint velocities are uniformly sampled from $[-0.005, 0.005]$ rad/s. The target joint positions are uniformly sampled from $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ rad for joint 1 and $[-\pi + 0.15, \pi - 0.15]$ rad for joint 2. During evaluations, the initial joint state is always set to the first state in the reference trajectory (Eq. 44).

E.1.1   Observation Function

We define the state $\mathbf{x} \in \mathbb{R}^4 = [q_1\ q_2\ \dot{q}_1\ \dot{q}_2]$ and the action $\mathbf{u} = [u_1\ u_2]$. The observation function $\phi(\mathbf{x}) \in \mathbb{R}^{17}$ we use in RKL and KL is

$$\phi(\mathbf{x})^\top = \begin{bmatrix} \mathbf{q}^\top & \dot{\mathbf{q}}^\top & 1 & \text{poly}(\mathbf{x})^\top & \text{tri}(\mathbf{x})^\top \end{bmatrix}, \tag{41}$$

where

$$\text{poly}(\mathbf{x})^\top = \begin{bmatrix} q_1^2 & q_1 q_2 & q_2^2 & \dot{q}_1^2 & \dot{q}_1\dot{q}_2 & \dot{q}_2^2 \end{bmatrix}, \tag{42}$$

$$\text{tri}(\mathbf{x})^\top = \begin{bmatrix} \sin q_1 & \sin q_2 & \sin(q_1 + q_2) & \cos q_1 & \cos q_2 & \cos(q_1 + q_2) \end{bmatrix}. \tag{43}$$

E.1.2   Reference Trajectory

The reference tip position trajectory is defined by

$$p_x = 0.05\sin\left(\frac{4\pi}{T}t\right) + 0.1, p_y = 0.1\cos\left(\frac{2\pi}{T}t\right), t \in [0, T], \tag{44}$$

where T in the period. We choose $T = 5$ seconds. The reference joint position trajectory is calculated using inverse kinematics, and the reference joint velocity trajectory is obtained by finite difference.

E.1.3 KL and RKL Implementation

We implement KL and RKL in Python, using a single thread. Every 10 ms, we compute the optimal action, update the model (if using RKL), and simulate the system one step forward. We test six different initial dataset sizes (500 Demo, 500, 1,000, 2,000, 3,000, and 5,000 data points) and two types of MPC solver (MPC-SAC and LQR) for both KL and RKL. The 500 Demo initial dataset is generated by using a PD controller to control the joints, and other initial datasets are generated by applying random actions uniformly sampled from the action space. For each of the resulting 24 policies, we perform 50 trials. The results are shown in Table 1 in the main paper and visualized in Fig. 7.

E.1.4 NN-MPPI Training

We use the NN-MPPI implementation from MaxDiffRL [47]. We train 15 policies using five random seeds (555, 666, 777, 888, and 999) and three different total numbers of steps (5,000, 10,000, and 15,000). For each trained policy, 50 trials are performed. The average results across all 5 random seeds are shown in Table 1 in the main paper for each "Dataset size" and visualized in Fig. 8. The parameters for training and the MPPI controller are shown in Table 5.

E.1.5 RL-SAC Training

We use the RL-SAC implementation from stable-baselines3 [48]. Training is performed on 32 independently and randomly initialized environments with different seeds using SubprocVecEnv. The evaluation results are shown in Table 1 in the main paper and visualized in Fig. 9. The training parameters are detailed in Table 6.

E.1.6 REDQ Training

We use the open-source code provided by the authors of REDQ [42]. The evaluation results are shown in Table 1 in the main paper and in Fig. 10. The training parameters are detailed in Table 7.

E.2 Soft Stewart Platform

Figure 4: The Soft Stewart Platform

The Soft Stewart Platform (SSP) is a six-DoF parallel manipulator inspired by the traditional Stewart platform design (Fig. 4). It is actuated by six Handed Shearing Auxetics (HSAs) [40], which are manufactured using fused deposition modeling (FDM) with TPU 95A filament on a Bambu X1-Carbon printer. Each actuator is anchored to the top of the platform and driven at the opposite end by a DYNAMIXEL XM430-W350-T servo motor. Unlike a conventional Stewart platform, this design eliminates joints at the ends of the struts. Instead, the compliance of the HSAs provides the kinematic behavior typically achieved with such joints. The platform's top surface is a hexagonal piece of acrylic, with each side measuring 24 cm, sanded to reduce friction. Surrounding walls and edges, made from 3D-printed polylactic acid (PLA), not only prevent the puck from falling off but also allow intermittent mechanical contact.

State information is obtained using two AprilTags mounted on the top and bottom of the platform. A Logitech BRIO webcam positioned above the setup tracks these tags while simultaneously detecting

Horizon	160 ms
LQR $\mathbf{q}$ weight	200.0
LQR $\dot{\mathbf{q}}$ weight	30.0
LQR observations weight	1.0
LQR $\mathbf{u}$ weights	0.001
LQR terminal weights	0.0
MPC-SAC $\mathbf{u}$ weights	1e10

Table 3: KL-SAC and RKL-SAC on the Planar Two-link Arm

Horizon	500 ms
LQR $\mathbf{p}$ weight	200.0
LQR $\dot{\mathbf{q}}$ weight	5.0
LQR observations weight	1.0
LQR $\mathbf{u}$ weights	0.001
LQR terminal weights	0.0

Table 4: KL-LQR and RKL-LQR Parameters on the Planar Two-link Arm

Horizon	300 ms
Number of samples	2000
Temperature	0.5
Gaussian noise std dev	0.5
Model learning rate	3e-4
Model training iterations	6
Model training batch size	128

Table 5: NN-MPPI Parameters on the Planar Two-link Arm

Batch size	256
Learning rate	3e-4
Buffer size	1e6
Learning start	1e4
Discount factor	0.99
Training frequency	1
Target network update rate	0.005

Table 6: RL-SAC Training Parameters on the Planar Two-link Arm

Batch size	256
Learning rate	3e-4
Learning start	2000
Buffer size	1e6
Discount factor	0.99
Polyak averaging factor	0.995
UTD ratio	20
#Q-functios	10
#Q-values when forming the target	2
Q-targets mode	min
Policy update delay	20

Table 7: REDQ Training Parameters on the Planar Two-link Arm

the balancing puck through a Hough circles algorithm. Using ray-tracing techniques, the system calculates the puck's position relative to the platform's center. The combined state of the platform-puck system is computed at approximately 60 Hz, with motor commands issued at 50 Hz.

E.2.1 NN-MPPI Implementation

Model-predictive path integral control (MPPI) is a model predictive control (MPC) algorithm [49], and NN-MPPI refers to the model-based reinforcement learning framework that uses MPPI alongside neural networks as dynamics models [11]. This was implemented and benchmarked on the Soft Stewart Platform in [43]. This approach uses an approximate geometric model of the SSP as if it were rigid, and learns residuals over this model to accomodate the platform's nonlinear actuators. The observation space is defined as $\mathbf{o} \in \mathbb{R}^9 = [x, y, \dot{x}, \dot{y}, \phi, \theta, \psi, X, Y]$. This vector contains $[x, y, \dot{x}, \dot{y}]$ the position and velocity of the puck, $[\phi, \theta, \psi]$ the Euler angles of the top platform surface relative to the base, and $[X, Y]$ the coordinates of the balancing setpoint. Actions are computed at 15 Hz.

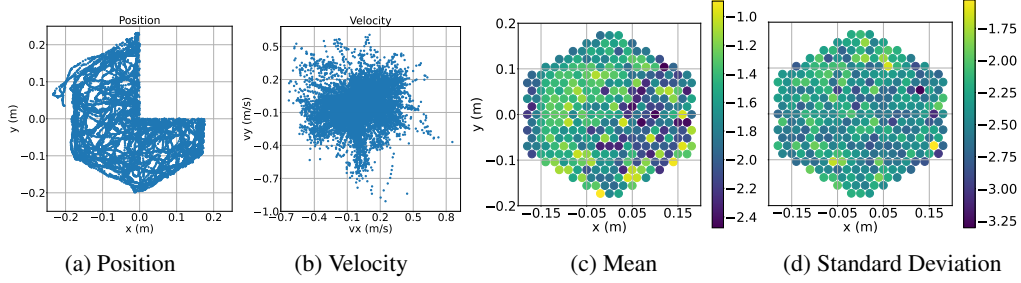(a) Position   (b) Velocity   (c) Mean   (d) Standard Deviation

Figure 5: The distribution of data in the cropped 4-minute initial dataset, as well as the mean and standard deviation of the absolute errors (scaled by a base-10 logarithm) computed over the last 5 seconds of each RKL-SAC trial using the cropped dataset. The mean error across the 293 test points is 0.03 m, with an average standard deviation of 0.0237 m.
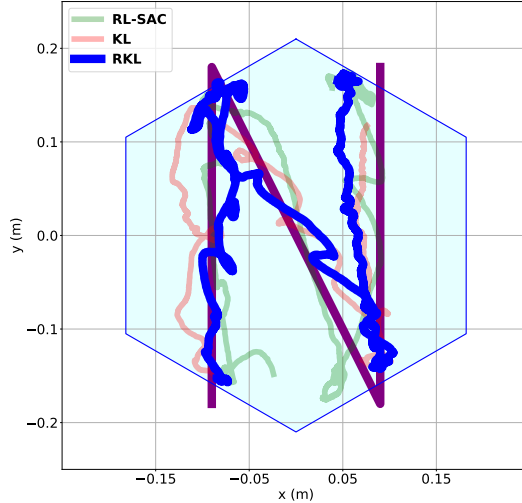

Figure 6: Tracking a reference trajectory containing contacts. The start, end and both corners of the reference trajectory are outside the platform boundary. The results shown are the best trials for each method (videos available). The cyan region denotes the platform area, which is slightly smaller than its actual size due to Apriltag measurement accuracy.

E.2.2   KL and RKL Implementation

State feedback: To model the system, we define the state $\mathbf{x} \in \mathbb{R}^4 = [p_x, p_y, v_x, v_y]$, which includes the position and velocity of the puck, and the control input $\mathbf{u} \in \mathbb{R}^6$, representing the angles of the six servos. The puck's current position is determined using a computer vision algorithm, and a Moving Average Filter (MAF) with a window size of approximately 65 ms is applied to the raw position feedback. The velocity is then estimated from the filtered positions using finite differences.

Initial dataset: The initial dataset is collected by a human operator using a SpaceMouse Compact and an inverse kinematic approximation that models the platform as if it was rigid. The purpose of this dataset is to create a reasonable initial model, thereby mitigating numerical issues, so no specific control objectives are provided during data collection. The same initial dataset is used for all experiments, and the distribution of the puck's positions in this dataset is depicted in Fig. 11 and 12.

Collecting raw state and control input data exhibits fluctuations and occasional missing values. To address this, linear interpolation is applied to align the state and input data to a consistent frequency. The processed initial dataset has a time step length of 10 ms. Subsequently, EDMD is performed on this dataset to obtain the initial Koopman model.

C++ implementation: The controller is implemented in C++ using a multi-threaded architecture. Four threads are employed:

- **Feedback thread** receives feedback on the puck's states.
- **Command thread** sends commands to the servos.
- **Control thread** solves the optimal control problem.
- **Update thread** updates the Koopman model with the latest data.

All threads run at a frequency of 50 Hz, constrained by the raw position feedback rate. However, in our experiments, MPC-SAC typically solves the control problem within 15 ms, and the model update is completed within 10 ms. In addition, an MAF is applied to the raw solutions before being sent to the servos.

E.2.3   KL and RKL Observation Functions

We use two types of observation functions in our experiments on the Soft Stewart Platform: a 28-dimensional polynomial basis function up to the third degree and a 117-dimensional Gaussian Radial Basis Function (RBF).

The 28-dimensional polynomial basis function is defined as

$$\boldsymbol{\phi}\left(\mathbf{x}\right)^{\top} = \begin{bmatrix} \mathbf{x}^{\top} & x_1^2 & x_1 x_2 & \cdots & x_1 x_2 x_3 \end{bmatrix}, \tag{45}$$

while the 117-dimensional Gaussian RBF is expressed as

$$\boldsymbol{\phi}\left(\mathbf{x}\right)^{\top} = \begin{bmatrix} \mathbf{x}^{\top} & \phi_1\left(\mathbf{x}\right) & \phi_2\left(\mathbf{x}\right) & \cdots & \phi_{117}\left(\mathbf{x}\right) \end{bmatrix}, \tag{46}$$

where each $\phi_i\left(\mathbf{x}\right)$ is defined by

$$\phi_i\left(\mathbf{x}\right) = \exp\left(-\epsilon \left\|\mathbf{x} - \mathbf{c}_i\right\|_2^2\right). \tag{47}$$

To define the centers $\mathbf{c}_i \in \mathbb{R}^4$, we follow the same methodology used to generate the goal points for the experiments. For the position component of $\mathbf{c}_i$, 117 uniformly distributed points are used, with a distance of 3.5 cm between each point and its six adjacent points. The sampling area for these points is slightly larger than the platform to ensure that positions near the boundaries can excite a sufficient number of RBFs, similar to positions within the platform. For the velocity component, k-means++ clustering is performed on the initial dataset to identify suitable velocity centers.

E.2.4   RL-SAC Training

We choose Soft Actor-Critic (RL-SAC) [7] as a benchmark for learned control. RL-SAC is an off-policy model-free maximum entropy deep reinforcement learning framework and is widely used for continuous control tasks. We train models in a single deployment—that is, non-episodically without environmental resets. We allow RL-SAC to train for 100,000 environment steps at a frequency of 10 Hz. To increase learning stability, we use a curriculum learning procedure that uniformly samples a new balance setpoint every 20 seconds, within a radius of the platform center. This radius is initialized to 3 cm and over the first half of the training procedure, it linearly expands to cover the entire platform. We retain the same input-output relationships as KL and RKL, with the addition of the current setpoint coordinates in the state vector $\mathbf{x}$. Five random seeds of RL-SAC are trained and evaluated. RL-SAC training does not have access to the initial dataset in Sec. F.

E.2.5   Parameters

The parameters for puck balancing using KL and RKL with the 28-dimensional polynomial basis function are shown in Table 8. The parameters for puck balancing using KL and RKL with the 117-dimensional Gaussian RBF are shown in Table 9. The parameters for trajectory tracking using KL and RKL with the 28-dimensional polynomial basis function are shown in Table 10. The parameters for RL-SAC training are shown in Table 11. The parameters for NN-MPPI training are shown in Table 12.
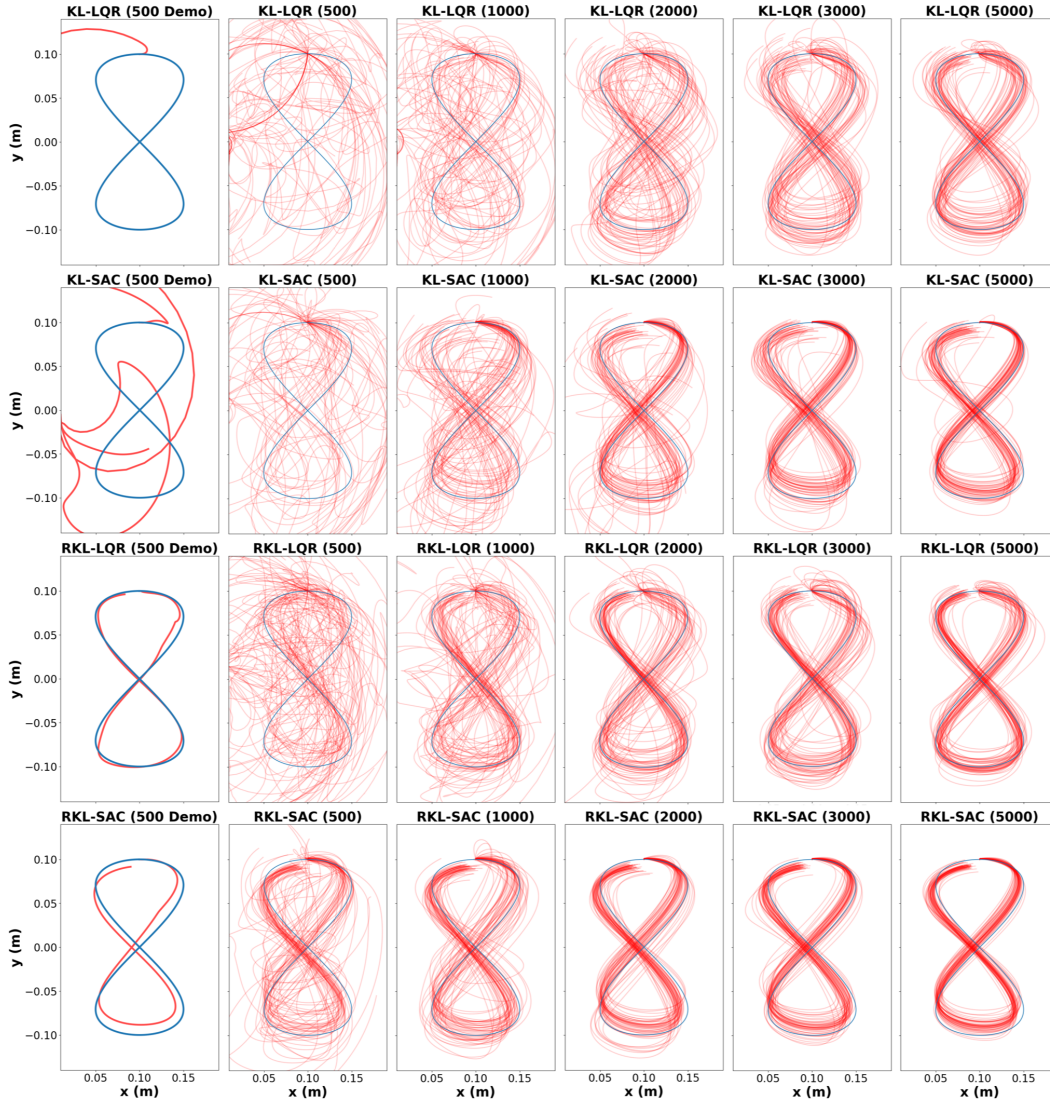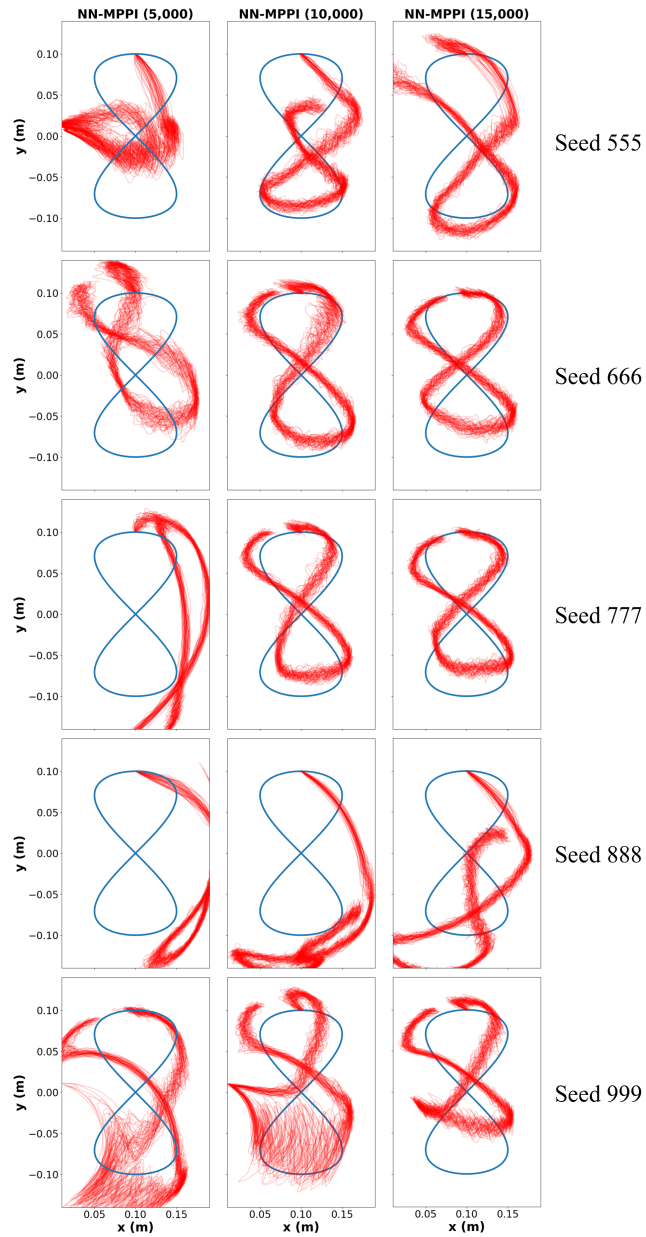
Figure 7: The results of tracking the reference tip position trajectory on the simulated planar two-link arm using KL and RKL with different initial dataset size. Since the generation of the dataset is random, except for the 500-demo dataset, 50 trials are conducted for each plot except for the first column. The red trajectory indicates the actual tip position, and the blue trajectory is the reference. By observing each column, it is obvious that RKL consistently demonstrates superior performance compared to KL. Furthermore, methods based on MPC-SAC outperform those based on LQR.

Figure 8: The results of tracking the reference tip position trajectory on the simulated planar two-link arm using policies trained by NN-MPPI. Each row corresponds to a distinct, fixed random seed, while each column corresponds to a different training steps. The red trajectory indicates the actual tip position, and the blue trajectory is the reference. Trajectories generated by NN-MPPI exhibit significant jitter, and the performance consistency across different random seeds is low.

Horizon	160 ms
LQR $\mathbf{p}$ weight	17.0
LQR $\mathbf{v}$ weight	1.0
LQR observations weight	0.0
LQR $\mathbf{u}$ weights	1.5
LQR terminal weights	0.0
MPC-SAC $\mathbf{u}$ weights	0.002
$\mathbf{u}$ MAF window size	120 ms

Table 8: Puck Balancing using Polynomial Observables

Horizon	160 ms
LQR $\mathbf{p}$ weight	17.0
LQR $\mathbf{v}$ weight	1.0
LQR observations weight	0.0
LQR $\mathbf{u}$ weights	0.001
LQR terminal weights	0.0
MPC-SAC $\mathbf{u}$ weights	4.25e-5
$\mathbf{u}$ MAF window size	120 ms

Table 9: Puck Balancing using Gaussain RBF Observables

Horizon	200 ms
LQR $\mathbf{p}$ weight	17.0
LQR $\mathbf{v}$ weight	1.5
LQR observations weight	0.0
LQR $\mathbf{u}$ weights	1.5
LQR terminal weights	0.0
MPC-SAC $\mathbf{u}$ weights	0.0015
$\mathbf{u}$ MAF window size	120 ms

Table 10: Puck Trajectory Tracking using Polynomial Observables

State Dim.	6
Action Dim.	6
Learning Rate	0.0003
Batch Size	128
Policy Network Dim.	[256]
Discount	0.99
Smoothing Coeff.	0.005
Reward Scale	0.1

Table 11: RL-SAC Parameters

State Dim.	9
Action Dim.	2
Horizon	20 steps
Learning Rate	0.0003
Batch Size	128
Network Dim.	[200, 200]
Discount	0.95
Samples	4096
Temperature λ	0.1

Table 12: NN-MPPI Parameters

F   Initial Dataset for SSP Experiments

We use the same initial dataset for all experiments on the Soft Stewart Platform. The distribution of samples are shown in Fig. 11 and 12.
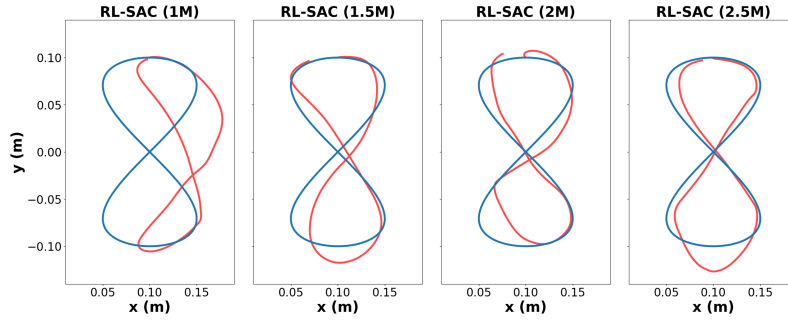
Figure 9: The results of tracking the reference tip position trajectory on the simulated planar two-link arm using policies trained by RL-SAC, with training steps 1 million to 2.5 million. The red trajectory indicates the actual tip position, and the blue trajectory is the reference. Despite the ability of RL-SAC to achieve adequate control, its sample efficiency is notably low.
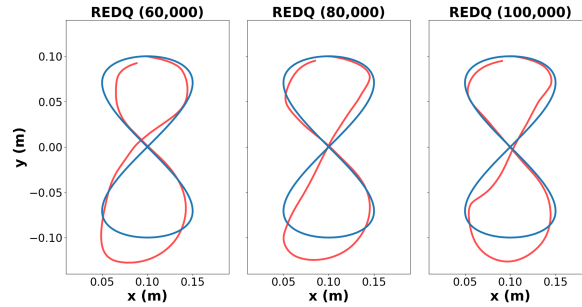

Figure 10: The results of tracking the reference tip position trajectory on the simulated planar two-link arm using policies trained by REDQ, with training steps 60K, 80K, and 100K. The red trajectory indicates the actual tip position, and the blue trajectory is the reference. As [42] states, REDQ does have high sample efficiency compared to other model-free RL methods, but RKL still significantly outperforms it.
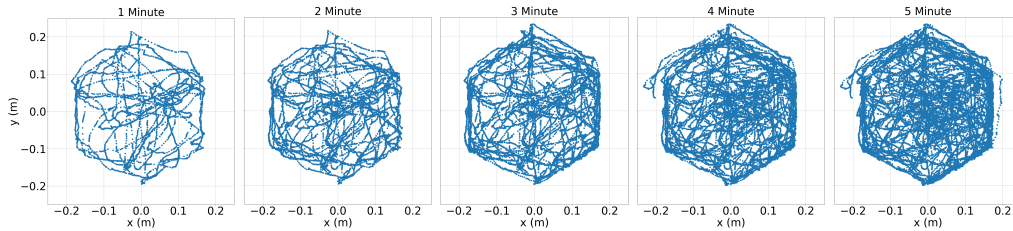

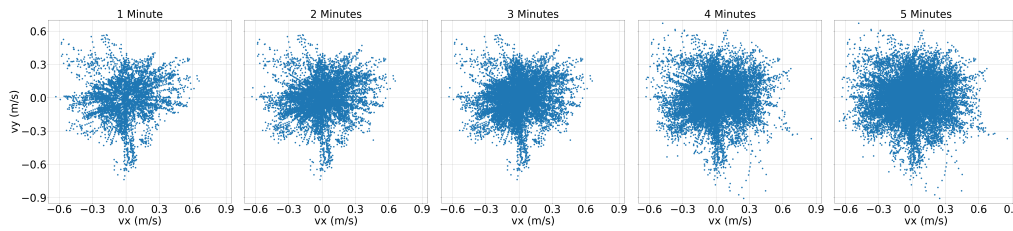Figure 11: The distribution of the puck's positions (m) in the initial dataset.

Figure 12: The distribution of the puck's velocities (m/s) in the initial dataset.