JDMD: Extended Dynamic Mode Decomposition with Jacobian Residual Penalization for Learning Bilinear, Control-affine Koopman Models

Brian E. Jackson
Robotics Institute
Carnegie Mellon University
brianjackson@cmu.edu

Jeong Hun Lee
Robotics Institute
Carnegie Mellon University
jeonghunlee@cmu.edu

Kevin Tracy
Robotics Institute
Carnegie Mellon University
ktracy@cmu.edu

Zachary Manchester
Robotics Institute
Carnegie Mellon University
zacm@cmu.edu

Abstract: Combining the benefits of data-driven and model-based control techniques for nonlinear dynamics systems has many potential advantages. Compared to purely data-driven approaches, this could include benefits such as increasing sampling efficiency, reducing training time, or increasing generalization. For purely model-based approaches such as model-predictive control, this could increase robustness to model uncertainty or improve overall performance by adapting the controller based on observations from the real system. We present a novel extension to previous data-driven approaches based on Koopman operator theory that combines many of the benefits of model-based and data-driven approaches. By including derivative information from an approximate analytical model, we show that we can quickly learn an efficient bilinear representation of the system dynamics, that, when combined with a novel linear MPC method that projects the dynamics into the original state space, offers performance improvements over nominal linear MPC with just a few training trajectories. We demonstrate increased sampling efficiency, increased generalization, and robustness to training hyperparameters compared to previous approaches

1 Introduction

Controlling complex, underactuated, and highly nonlinear autonomous systems remains an active area of research in robotics, despite decades of previous work exploring effective algorithms and the development of substantial theoretical analysis. Classical approaches typically rely on local linear approximations of the nonlinear system, which are then used in any of a multitude of linear control techniques, such as PID, pole placement, Bode analysis, H-infinity, LQR, or linear MPC. These approaches only work well if the states of the system always remain close to the equilibrium point or reference trajectory about which the system was linearized. The region for which these linearizations remain valid can be extremely small for highly nonlinear systems. Alternatively, model-based methods for nonlinear optimal control have shown great success, as long as the model is well known and an accurate estimate of the global state can be provided. These model-based techniques leverage decades of insight into dynamical systems and have demonstrated incredible performance on complicated autonomous systems [1, 2, 3, 4] . On the other hand, data-driven techniques such as reinforcement learning have received tremendous attention over the last decade and have begun to demonstrate impressive performance and robustness for complicated robotic systems in unstructured environments [5, 6, 7]. While these approaches are attractive since they require little to no previous knowledge about the system, they often require large amounts of data and fail to generalize outside of the domain or task on which they were "trained."

In this work we propose a novel method that combines the benefits of model-based and data-driven methods, based on recent work applying Koopman Operator Theory to controlled dynamical systems [8, 9, 10, 11, 12]. By leveraging data collected from an unknown dynamical system along with derivative information from an approximate analytical model, we can efficiently learn a bilinear representation of the system dynamics that performs well when used in traditional model-based control techniques such as linear MPC. By leveraging information from an analytical model, we can dramatically reduce the number of samples required to learn a good approximation of the true nonlinear dynamics. We also show the effectiveness of linear MPC on these systems when the learned bilinear system is linearized and projected back into the original state space. The result is a fast, robust, and sample-efficient pipeline for quickly learning a model that beats previous Koopman-based linear MPC approaches as well as purely model-based linear MPC controllers that do not leverage data collected from the actual system. To efficiently learn these bilinear representations, we also propose a numerical technique that allows for large systems to be trained while limiting the peak memory required to solve the least-squares problem.

In summary, our contributions are:

- A novel extension to extended dynamic mode decomposition (eDMD) that incorporates gradient information from an approximate analytic model;
- a simple linear MPC control technique for learned bilinear control systems that is computationally efficient online, which, when combined with the proposed extension to eDMD, requires extremely little training data to get a good control policy; and
- a recursive, batch QR algorithm solving the least-squares problems that arise when learning bilinear dynamical systems using eDMD.

The paper is organized as follows: in Section 2 we give some background on the application of Koopman operator theory to controlled dynamical systems and review some related works. Section 3 describes the proposed algorithm for combining data-driven and model-based approaches, along with the numerical method for solving the resulting large and sparse linear least-squares problems. Section 4 provides extensive numerical analysis of the proposed algorithm, applied to a simulated cartpole and planar quadrotor model, both subject to significant model mismatch. In Section 5 we discuss the limitations of the method, and finish with some concluding thoughts in Section 6.

2 Background and Related Work

The theoretical underpinnings of the Koopman operator and its application to dynamical systems has been extensively studied, especially within the last decade [13, 14, 9, 15]. Rather than describe the theory in detail, we highlight the key concepts employed by the current work, and defer the motivated reader to the existing literature on Koopman theory.

We start by assuming we have some discrete approximation a of controlled nonlinear, time-dynamical system whose underlying continuous dynamics are Lipchitz continuous:

$$x^+ = f(x, u) \tag{1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^{N_x}$ is the state vector, $u_k \in \mathbb{R}^{N_u}$ is the control vector, and x^+ is the state at the next time step. This discrete approximation can be obtained for any continuous-time, smooth dynamical system in many ways, including implicit and explicit Runge-Kutta methods, or by solving the Discrete Euler-Lagrange equations [16, 17, 18].

The key idea behind the Koopman operator is that the nonlinear finite-dimensional discrete dynamics (1) can be represented by an infinite-dimensional *bilinear* system:

$$y^+ = Ay + Bu + \sum_{i=1}^{m} u_i C_i y = g(y, u) \tag{2}$$

where $y = \phi(x)$ is a nonlinear mapping from the finite-dimensional state space $\mathcal{X}$ to the (possibly) infinite-dimensional Hilbert space of *observables* $y \in \mathcal{Y}$. We also assume the inverse map is approximately linear: $x = Gy$. In practice, we approximate (1) by choosing ϕ to be some arbitrary finite set of nonlinear functions of the state variables, which in general include the states themselves such that the linear mapping $G \in \mathbb{R}^{N_x \times N_y}$ is exact. Intuitively, ϕ "lifts" our states into a higher dimensional space where the dynamics are approximately (bi)linear, effectively trading dimensionality for (bi)linearity. This idea should be both unsurprising and familiar to most roboticists, since similar techniques have already been employed in other forms, such as maximal-coordinate representations of rigid body dynamics [19, 16, 18], the "kernel trick" for state-vector machines [20], or the observation that solving large, sparse nonlinear optimization problems is often more effective than solving small, tightly-coupled dense problems TODO: add citations from the trajectory optimization literature.

The lifted bilinear system (2) can be easily learned from samples of the system dynamics (x_j^+, x_j, u_j) using extended Dynamic Mode Decomposition (eDMD) [15], which is just the application of linear least squares (LLS) to the lifted states. Details of this method will be covered in the next section where we introduce our adaptation of eDMD and present an effective numerical technique for solving the resulting LLS problems.

The current work is most closely related to recent work out of Caltech [11, 21, 22] which learn a bilinear model and apply nonlinear model-predictive control directly on the learned bilinear dynamics. Similarly, control Lyapunov functions were also recently used directly on the learned bilinear dynamics [23]. These works build on a foundational paper applying MPC to nonlinear systems using Koopman operators to learn a linear model directly [10].

3 EDMD with Jacobian Residual-Penalization

Existing Koopman-based approaches to learning dynamical systems only rely on samples of the unknown dynamics. Here we present a novel method for incorporating prior knowledge about the dynamics by adding derivative information of an approximate model into the data set to be learned via eDMD.

Given P samples of the dynamics (x_i^+, x_i, u_i), and an approximate discrete dynamics model

$$x^+ = \tilde{f}(x, u) \tag{3}$$

we can evaluate the Jacobians of our approximate model $\tilde{f}$ at each of the sample points: $\tilde{A}_i = \frac{\partial \tilde{f}}{\partial x}, \tilde{B}_i = \frac{\partial \tilde{f}}{\partial u}$. After choosing a nonlinear mapping $\phi : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_y}$ our goal is to find a bilinear dynamics model (2) that matches the Jacobians of our approximate model, while also matching our dynamics samples. If we define $\hat{A}_j \in \mathbb{R}^{N_x \times N_x}$ and $\hat{B}_j \in \mathbb{R}^{N_x \times N_u}$ to be the Jacobians of our bilinear dynamics model, projected back into the original state space (a formal definition of these terms will be provided in a few paragraphs), our objective is to find the matrices that parameterize our bilinear dynamics model, $A \in \mathbb{R}^{N_y \times N_y}, B \in \mathbb{R}^{N_y \times N_u}$, and $C_{1:m} \in \mathbb{R}^{N_u} \times \mathbb{R}^{N_y \times N_y}$, that minimize the following objective:

$$(1-\alpha) \sum_{j=1}^{P} \left\|\hat{y}_j - y_j^+\right\|_2^2 + \alpha \sum_{j=1}^{P} \left\|\hat{A}_j - \tilde{A}_j\right\|_2^2 + \left\|\hat{B}_j - \tilde{B}_j\right\|_2^2 \tag{4}$$

where $\hat{y}_j^+ = g(\phi(x_j), u_j)$ is the output of our bilinear dynamics model, and $y_j^+ = \phi(x_j^+)$ is the actual lifted state (i.e. observables) at the next time step. Note that $\hat{y}_j$, $\hat{A}_j$, and $\hat{B}_j$ are all implicitly functions of the model parameters A, B, and $C_{1:m}$ we're trying to learn.

While not immediately apparent, we can minimize (4) using linear least-squares, using techniques similar to those used previously in the literature [21].

To start, we combine all the data we're trying to learn into a single matrix:

$$E = [A \quad B \quad C_1 \quad \dots \quad C_m] \in \mathbb{R}^{N_y \times N_z}, \tag{5}$$

where $N_z = N_y + N_u + N_y \cdot N_u$. We now rewrite the terms in (4) in terms of E. By defining the vector

$$z = \begin{bmatrix} y^T & u^T & u_1 y^T & \dots & u_m y^T \end{bmatrix} \in \mathbb{R}^{N_z}, \tag{6}$$

we can write down the output of our bilinear dynamics (2) as

$$\hat{y}^+ = Ez. \tag{7}$$

The previously-mentioned projected Jacobians of our bilinear model, $\hat{A}$ and $\hat{B}$, are simply the Jacobians of the bilinear dynamics in terms of the original state. We obtain these dynamics by "lifting" the state via ϕ and then projecting back onto the original states using G:

$$x^+ = G\left(A\phi(x) + Bu + \sum_{i=1}^{m} u_i C_i \phi(x)\right) = \hat{f}(x,u) \tag{8}$$

Differentiating these dynamics gives us our projected Jacobians:

$$\hat{A}_j = \frac{\partial \hat{f}}{\partial x}(x_j, u_j) = G\left(A + \sum_{i=1}^{m} u_{j,i} C_i\right)\Phi(x_j) = GE\bar{A}(x_j,u_j) = GE\bar{A}_j \tag{9a}$$

$$\hat{B}_j = \frac{\partial \hat{f}}{\partial u}(x_j, u_j) = G\Big(B + [C_1 x_j \quad \dots \quad C_m x_j]\Big) = GE\bar{B}(x_j,u_j) = GE\bar{B}_j \tag{9b}$$

where $\Phi(x) = \partial\phi/\partial x$ is the Jacobian of the nonlinear map ϕ, and

$$\bar{A}(x,u) = \begin{bmatrix} I \\ 0 \\ u_1 I \\ u_2 I \\ \vdots \\ u_m I \end{bmatrix} \in \mathbb{R}^{N_z \times N_x}, \quad \bar{B}(x,u) = \begin{bmatrix} 0 \\ I \\ [x \; 0 \; \dots \; 0] \\ [0 \; x \; \dots \; 0] \\ \vdots \\ [0 \; 0 \; \dots \; x] \end{bmatrix} \in \mathbb{R}^{N_z \times N_u}. \tag{10}$$

Note we define $\bar{A}_j = \bar{A}(x_j,u_j)$, $\bar{B}_j = \bar{B}(x_j,u_j)$ to lighten the notation, but want to emphasize that these terms are all purely functions of the input data.

Substituting (7) and (9) into (4), we can rewrite our LLS problem as:

$$\underset{E}{\text{minimize}} \sum_{j=0}^{P} (1-\alpha)\left\|Ez_j - y_j^+\right\|_2^2 + \alpha\left\|GE\bar{A}_j - \tilde{A}_j\right\|_2^2 + \alpha\left\|GE\bar{B}_j - \tilde{B}_j\right\|_2^2 \tag{11}$$

which is equivalent to

$$\underset{E}{\text{minimize}} \; (1-\alpha)\left\|E\mathbf{Z_{1:P}} - \mathbf{Y^+_{1:P}}\right\|_2^2 + \alpha\left\|GE\bar{\mathbf{A}}_{\mathbf{1:P}} - \tilde{\mathbf{A}}_{\mathbf{1:P}}\right\|_2^2 + \alpha\left\|GE\bar{\mathbf{B}}_{\mathbf{1:P}} - \tilde{\mathbf{B}}_{\mathbf{1:P}}\right\|_2^2 \tag{12}$$

where $\mathbf{Z_{1:P}} \in \mathbb{R}^{N_z \times P} = [z_1 \; z_2 \; \dots \; z_P]$ horizontally concatenates all of the samples (equivalent definition for $\mathbf{Y^+_{1:P}} \in \mathbb{R}^{N_y \times P}$, $\bar{\mathbf{A}}_{\mathbf{1:P}} \in \mathbb{R}^{N_z \times N_x \cdot P}$, $\tilde{\mathbf{A}}_{\mathbf{1:P}} \in \mathbb{R}^{N_z \times N_x \cdot P}$, $\bar{\mathbf{B}}_{\mathbf{1:P}} \in \mathbb{R}^{N_z \times N_u \cdot P}$, and $\tilde{\mathbf{B}}_{\mathbf{1:P}} \in \mathbb{R}^{N_z \times N_u \cdot P}$).

We can rewrite (12) in standard form using the "vec trick"

$$\text{vec}(AXB) = (B^T \otimes A)\text{vec}(X) \tag{13}$$

where vec(A) stacks the columns of A into a single vector.

Setting E in (14) equal to X in (13), we get

$$\underset{E}{\text{minimize}} \left\| \begin{bmatrix} (1-\alpha)\cdot(\mathbf{Z_{1:P}})^T \otimes I_{N_y} \\ \alpha \cdot (\bar{\mathbf{A}}_{\mathbf{1:P}})^T \otimes G \\ \alpha \cdot (\bar{\mathbf{G}}_{\mathbf{1:P}})^T \otimes G \end{bmatrix} \text{vec}(E) + \begin{bmatrix} (1-\alpha)\cdot \text{vec}(\mathbf{Y^+_{1:P}}) \\ \alpha \cdot \text{vec}(\tilde{\mathbf{A}}_{\mathbf{1:P}}) \\ \alpha \cdot \text{vec}(\tilde{\mathbf{G}}_{\mathbf{1:P}}) \end{bmatrix} \right\|_2^2 \tag{14}$$

such that the matrix of coefficients has $(N_y + N_x^2 + N_x \cdot N_u) \cdot P$ rows and $N_y \cdot N_z$ columns. We obtain the data for our bilinear model (2) by solving this large, sparse linear least-squares problem.

3.1 Efficient Recursive Least Squares

In its canonical formulation, a linear least squares problem can be represented as the following unconstrained optimization problem:

$$\min_x \|Fx - d\|_2^2. \tag{15}$$

The solution to this problem is found by solving for the x in which the gradient of the objective function with respect to x is zero, also known as the normal equations:

$$(F^T F)x = F^T d, \tag{16}$$

For small to medium sized problems, this problem is most often solved with either a Cholesky or a QR decomposition. Unfortunately, for very large problems where storage size and numerical conditioning become a concern, forming and factorizing the required matrices can be intractable.

To deal with large problems like the one proposed in (14), a recursive method is used that processes rows of F and d sequentially in batches, avoiding the need for forming or factorizing the whole matrix. To do this, the rows of F and d will be divided up into batches in the following manner:

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{bmatrix}, \quad d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}. \tag{17}$$

The matrix $F^T F$ from (16) can then be represented as the following sum:

$$F^T F = F_1^T F_1 + F_2^T F_2 + \ldots + F_N^T F, \tag{18}$$

with the right-hand side vector in (16) expressed in a similar fashion:

$$F^T d = F_1^T d_1 + F_2^T d_2 + \ldots + F_N^T d_N. \tag{19}$$

The main idea of this recursive method is to maintain an upper-triangular "square root" factor U_i of the first i terms of the sum (18). Given the factorization U_i, we can calculate U_{k+1} using the QR decomposition, as shown in [24]:

$$U_{i+1} = \sqrt{U_i + F_{i+1}} = \mathrm{QR_R}\left(\begin{bmatrix} \sqrt{U_i} \\ \sqrt{F_{i+1}} \end{bmatrix} \right), \tag{20}$$

where $\mathrm{QR_R}$ returns the upper triangular matrix R from the QR decomposition.

We handle regularization of the normal equations, equivalent to adding L2 regularization to the original least squares problem, during the base case of our recursion. If we want to add an L2 regularization with weight ρ, we calculate U_1 as:

$$U_1 = \mathrm{QR_R}\left(\begin{bmatrix} \sqrt{F_1} \\ \sqrt{\rho I} \end{bmatrix} . \right), \tag{21}$$

The final algorithm for solving a least squares problem in a recursive-batch fashion is described in algorithm 1. This algorithm can be modified to handle L2 regularized least squares problems by simply replacing line 2 of algorithm 1 with $U \leftarrow \mathrm{QR_R}(\mathrm{vcat}(F_1, \sqrt{\rho} I))$, where ρ is the regularizer.

4 Results

The following sections provide various numerical analyses of the proposed algorithm. In lieu of an actual hardware experiment (left for future work), for each simulated system we specify two models: a *nominal* model which is a simplified model approximating the *simulated* model, which contains both parametric and non-parametric model error from the nominal model. This simulated model is used exclusively for simulating the system.

Algorithm 1 Recursive Batch Least Squares with QR

```
1: input $F$, $d$                                    ▷ problem data
2: $U \leftarrow \mathrm{QR_R}(\mathrm{vcat}(F_1, \sqrt{\rho} I))$    ▷ form initial upper-triangular square-root
3: $b \leftarrow F_1^T d_1$                          ▷ form initial right hand side vector
4: for $i = 2 : N$ do
5:     $U \leftarrow \mathrm{QR_R}(\mathrm{vcat}(U, F_i))$       ▷ update square-root with new batch
6:     $b \leftarrow b + F_i^T d_i$                  ▷ update right hand side with new batch
7: end for
8: output $x \leftarrow U^{-1} U^{-T} b$             ▷ forward and backwards substitution to solve for $x$
```

All models were trained by simulating the "real" system with an arbitrary controller to collect data in the region of the state space relevant to the task. A set of fixed-length trajectories were collected, each at a sample rate of 20 Hz. The bilinear eDMD model was trained using the same approach introduced by Folkestad and Burdick [21]. For the proposed jDMD method, the Jacobians of the nominal model were calculated at each of the sample points and the bilinear model was learned using the approach outlined in Section 3. All continuous dynamics were discretized with an explicit fourth-order Runge Kutta integrator. The code for the experiments is located at TODO: include after review.

We organize the following results section by topic, including examples from both the canonical cartpole system, as well as a planar quadrotor model derived from experimental data on hardware platforms in our lab.

The *simulated* cartpole model included a $\tanh$ model of Coulomb friction between the cart and the floor, viscous damping at both joints, and a control dead band that was not included in the *nominal* model. Additionally, the mass of the cart and pole model were altered by 20% and 25% with respect to the nominal model, respectively. The following nonlinear mapping was used when learning the bilinear models: $\phi(x) = [1, x, \sin(x), \cos(x), \sin(2x), \sin(4x), T_2(x), T_3(x), T_4(x)] \in \mathbb{R}^{33}$, where $T_i(x)$ is a Chebyshev polynomial of the first kind of order i. All reference trajectories for the swing up task were generated using ALTRO [24, 25].

For the planar quadrotor, the *simulated* model included aerodynamic drag terms not included in the *nominal* model, as well as parametric error of about 5% on the system properties (e.g. mass, rotor arm length, etc.). The model was trained using a nonlinear mapping of $\phi(x) = [1, x, \sin(x), \cos(x), \sin(2x), T_2(x),] \in \mathbb{R}^{TODO: get dimension}$.

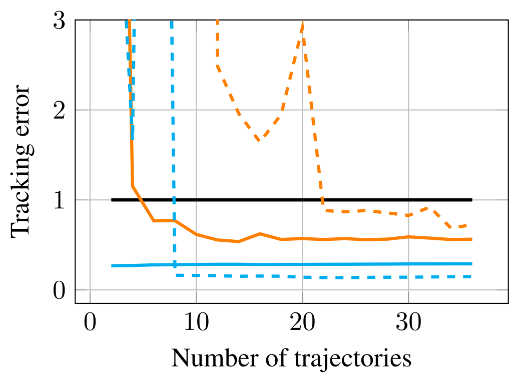
4.1 Sample Efficiency

We highlight the sample efficiency of the proposed algorithm in Figure 1a. For both the cartpole swing up and the quadrotor trajectory tracking tasks, the proposed method achieves better tracking than the nominal MPC controller after just a few training trajectories. In both cases, the classic eDMD approach never achieves the same level of performance as the proposed approach. The lack of continued progress with increasing samples is likely due to a lack of sufficient variety in the training data: after 30-40 training trajectories both methods have effectively learned as much as they can from the distribution from which the training data was sampled.

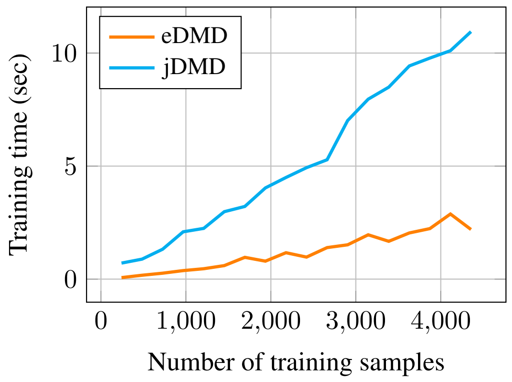
The training time versus number of training samples is shown in Figure 1b for the cartpole swing up task. While the proposed approach naturally takes longer since it includes much more information per sample (adding $N_x + N_u + 1$ rows for every sample), the complexity is approximately linear and the solve times are on the order of seconds for these simple systems TODO: add lines for quadrotor training.

4.2 Generalization

We demonstrate the generalizability of the proposed method to tasks outside of its training domain in Figure 2. In both the quadrotor stabilization (Figure **??**) and trajectory tracking (Figure 2a) tasks,

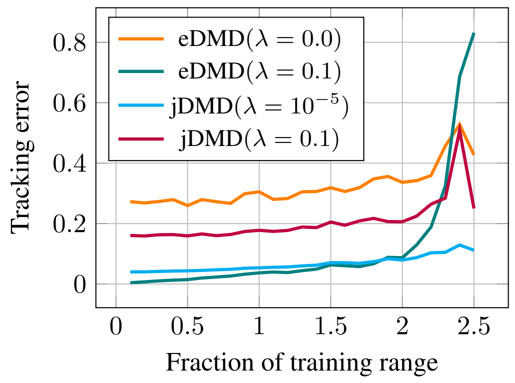


(a) MPC tracking error vs training samples for both the cartpole (solid lines) and planar quadrotor (dashed lines). Tracking error is defined as the average L2 error over all the test trajectories between the reference and simulated trajectories, and is normalized by the error of the nominal MPC controller (black horizontal line).

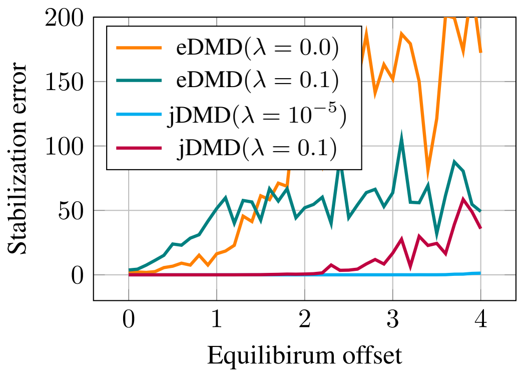


(b) Training time for cartpole models as a function of training samples, using a preliminary implementation of the algorithm described in Section 3.1. The training time complexity is approximately linear.

Figure 1: Effect of increased training data on controller performance (left) and training times (right) for the eDMD (orange) and jDMD (cyan) algorithms.



(a) Tracking error for the quadrotor MPC reference trajectory tracking task.



(b) LQR stabilization error over increasing equilibrium offset

Figure 2: Generalizability with respect to initial conditions sampled outside of the training domain. The initial conditions are sampled from a uniform distribution, whose limits are determined by a scaling of the limits used for the training distribution. A training range fraction greater than 1 indicates the distribution range is beyond that used to generate the training trajectories.

we trained the models by sampling uniformly from a given window of offsets, centered about the origin. To test the generalizability of the methods we increased the relative size of the window from which the test data was sampled, e.g. if the initial lateral position was trained on data in the interval $[-1.5, +1.5]$, we sampled the test initial condition from the window $[-\gamma 1.5, +\gamma 1.5]$. As shown in the results, while the performance of the proposed algorithm remains relatively constant even when $\gamma = 2.5$, whereas the classic eDMD approach looses performance and fails to generalize at all for the stabilization task using and LQR controller (like due to poor derivative information), and up to $\gamma = 2$ for the tracking task using a linear MPC controller.

In Figure 2b we show the effect of changing the equilibrium position for the planar quadrotor, but keeping the delta initial conditions within the training window. As shown, eDMD doesn't generalize to other equilibrium points, despite the fact that the underlying dynamics are invariant to the equilib-

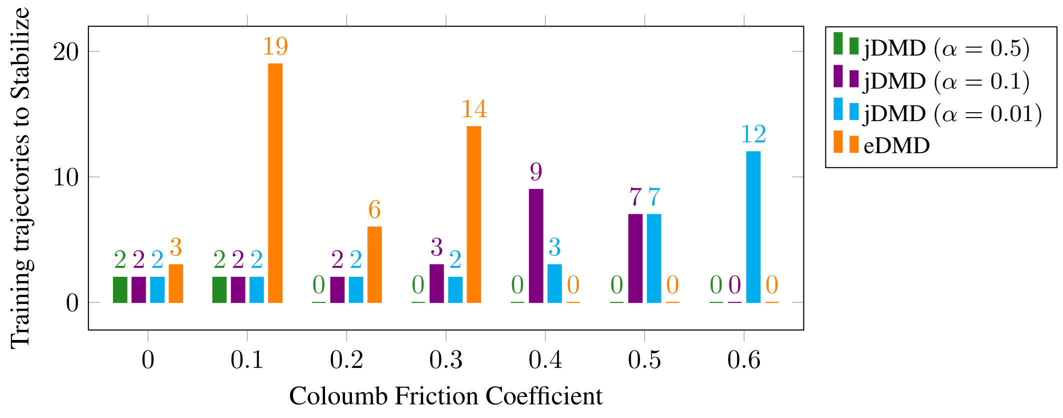


Figure 3: Effect of increasing model mismatch. Displays the number of training trajectories required to consistently stabilize the cartpole system, given an increasing friction coefficient, which the nominal model does not include at all. An entry of 0 signifies that a stabilizing controller wasn't found with less than 100 training trajectories. The nominal MPC controller failed to stabilize once the friction coefficient went above 0.1.

rium position. Our proposed approach, however, easily learns this from the derivative information provided by the nominal model.

4.3 Lifted versus Projected MPC

Table 1 also highlights the sample efficiency of the proposed method, while also comparing to the more typical approach of applying the MPC policy in the lifted state space [10, 21, 23]. As shown, the proposed method of applying linear MPC to the projected linearization is much more sample efficient than trying to apply it in the lifted state space. This approach is also advantageous because it reduces the solve time of the linear MPC policy, is more numerically robust (we found the lifted MPC policies tended to suffer from numerical issues when using Riccati recursion to solve for long time horizons), and it is straightforward to apply additional constraints to the original state variables.

MPC	**eDMD**	**jDMD**
Lifted	17	15
Projected	18	**2**

Table 1: Training trajectories required to beat nominal MPC

4.4 Sensitivity to Model Mismatch

While we've introduced a significant mount of model mismatch in all of the examples so far, a natural argument against model-based methods is that they're only as good as your model is at capturing the salient dynamics of the system. We investigated the effect of increasing model mismatch by incrementally increasing the Coulomb friction coefficient between the cart and the floor for the cartpole stabilization task. The results are shown in Figure 3. As expected, the number of training trajectories required to find a good stabilizing controller increases for the proposed approach, as long as we set the α value low enough, which intuitively corresponds to a decreased confidence in the nominal model. The samples required by eDMD varied widely, and was unable to find a good enough model above friction values of 0.4. While this could likely be remedied by adjusting the nonlinear mapping ϕ, the proposed approach works well with the given bases.

5 Limitations

As with most data-driven techniques, it is hard to definitely declare that the proposed method will increase performance in all cases. It is possible that having an extremely poor analytical model may hurt rather than help the training process. However, we found that even when the α parameter is extremely small (placing little weight on the Jacobians during the learning process), it still dramatically improves the sample efficiency. It is also quite possible that the performance gaps between eDMD and jDMD shown here can be reduced through better selection of basis functions and better training data sets; however, given that the proposed approach converges to eDMD as $\alpha \to 0$, we see no reason to not adopt the proposed methodology as simply tune α based on the confidence of the model and the quantity (and quality) of training data.

6 Conclusion and Future Work

We have presented a simple but powerful extension to eDMD, a model-based method for learning a bilinear representation of arbitrary dynamical systems, that incorporates derivative information from an analytical mode. When combined with a simple linear MPC policy that projects the learned dynamics back into the original state space, we have shown that the resulting pipeline can dramatically increase sample efficiency, often improving over a nominal MPC policy with just a few sample trajectories. Substantial areas for future work remain: most notably testing the proposed pipeline on hardware. Additional directions include lifelong learning or adaptive control applications, residual dynamics learning, as well as the development of specialized numerical methods for solving nonlinear optimal control problems using the learned bilinear dynamics.

References

[1] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli. An efficient optimal planning and control framework for quadrupedal locomotion. In *2017 {IEEE} International Conference on Robotics and Automation ({ICRA})*, pages 93–100. doi:10.1109/ICRA.2017.7989016.

[2] S. Kuindersma, F. Permenter, and R. Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. pages 2589–2594. ISSN 9781479936854. doi:10.1109/ICRA.2014.6907230.

[3] M. Bjelonic, R. Grandia, O. Harley, C. Galliard, S. Zimmermann, and M. Hutter. Whole-Body MPC and Online Gait Sequence Generation for Wheeled-Legged Robots. pages 8388–8395. ISSN 9781665417143. doi:10.1109/IROS51168.2021.9636371.

[4] J. K. Subosits and J. C. Gerdes. From the racetrack to the road: Real-time trajectory replanning for autonomous driving. 4(2):309–320. doi:10.1109/TIV.2019.2904390.

[5] N. Karnchanachari, M. I. Valls, S. David Hoeller, and M. Hutter. Practical Reinforcement Learning For MPC: Learning from sparse objectives in under an hour on a real robot. pages 1–14. doi:10.3929/ETHZ-B-000404690. URL `https://doi.org/10.3929/ethz-b-000404690`.

[6] D. . Hoeller, F. . Farshidian, M. Hutter, F. Farshidian, and D. Hoeller. Deep Value Model Predictive Control. 100:990–1004. doi:10.3929/ETHZ-B-000368961. URL `https://doi.org/10.3929/ethz-b-000368961`.

[7] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath. Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots. 2021-May:2811–2817. ISSN 9781728190778. doi:10.1109/ICRA48506.2021.9560769.

[8] 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. doi:10.48550/arxiv.2201.07601. URL `https://arxiv.org/abs/2201.07601v1`.

[9] D. Bruder, X. Fu, and R. Vasudevan. Advantages of Bilinear Koopman Realizations for the Modeling and Control of Systems with Unknown Dynamics. 6(3):4369–4376. doi:10.1109/LRA.2021.3068117.

[10] M. Korda and I. Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. 93:149–160. doi:10.1016/j.automatica.2018.03.046. URL https://doi.org/10.1016/j.automatica.2018.03.046.

[11] C. Folkestad, D. Pastor, and J. W. Burdick. Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Multirotor Landing. pages 9216–9222. ISSN 9781728173955. doi:10.1109/ICRA40945.2020.9197510.

[12] H. J. Suh and R. Tedrake. The Surprising Effectiveness of Linear Models for Visual Foresight in Object Pile Manipulation. 17:347–363. doi:10.48550/arxiv.2002.09093. URL https://arxiv.org/abs/2002.09093v3.

[13] SINDy with Control: A Tutorial. URL https://github.com/urban-fasel/SEIR.

[14] J. L. Proctor, S. L. Brunton, and J. Nathan Kutz. Generalizing koopman theory to allow for inputs and control. 17(1):909–930. doi:10.1137/16M1062296. URL http://www.siam.org/journals/siads/17-1/M106229.html.

[15] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. 25(6):1307–1346. doi:10.1007/S00332-015-9258-5/FIGURES/14. URL https://link.springer.com/article/10.1007/s00332-015-9258-5.

[16] J. Brüdigam and Z. Manchester. Linear-Time Variational Integrators in Maximal Coordinates. 17:194–209, . doi:10.1007/978-3-030-66723-8_12/FIGURES/8. URL https://link.springer.com/chapter/10.1007/978-3-030-66723-8_12.

[17] J. Brüdigam and Z. Manchester. Linear-Quadratic Optimal Control in Maximal Coordinates. 2021-May:3546–3552, . ISSN 9781728190778. doi:10.1109/ICRA48506.2021.9561871.

[18] T. A. Howell, S. Le Cleac', J. Z. Kolter, M. Schwager, and Z. Manchester. Dojo: A Differentiable Simulator for Robotics.

[19] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–146. ACM.

[20] M. Hofmann. Support Vector Machines-Kernels and the Kernel Trick An elaboration for the Hauptseminar "Reading Club: Support Vector Machines".

[21] C. Folkestad and J. W. Burdick. Koopman NMPC: Koopman-based Learning and Nonlinear Model Predictive Control of Control-affine Systems. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2021-May, pages 7350–7356. Institute of Electrical and Electronics Engineers Inc. ISBN 978-1-72819-077-8. doi:10.1109/ICRA48506.2021.9562002.

[22] C. Folkestad, S. X. Wei, and J. W. Burdick. Quadrotor Trajectory Tracking with Learned Dynamics: Joint Koopman-based Learning of System Models and Function Dictionaries. URL http://arxiv.org/abs/2110.10341.

[23] A. Narasingam, J. Sang, and I. Kwon. Data-driven feedback stabilization of nonlinear systems: Koopman-based model predictive control. pages 1–12.

[24] T. A. Howell, B. E. Jackson, and Z. Manchester. ALTRO: A Fast Solver for Constrained Trajectory Optimization. pages 7674–7679. ISSN 9781728140049. doi:10.1109/IROS40897.2019.8967788.

[25] B. E. Jackson, T. Punnoose, D. Neamati, K. Tracy, R. Jitosho, and Z. Manchester. ALTRO-C: A Fast Solver for Conic Model-Predictive Control; ALTRO-C: A Fast Solver for Conic Model-Predictive Control. ISSN 9781728190778. doi:10.1109/ICRA48506.2021.9561438. URL https://github.com/.