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

TODO: Add abstract

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 effiectiveness 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 contollers 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 appliciation 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 mistmach. In Section 5 we discuss the limitations of the method, and finish with some concluding thoughts in Section 6.

2 Background and Related Work

2.1 The Koopman Operator

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 Lipschitz 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 roboticsts, 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.

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 parameterizing 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 $\hat{y}_j^+ = g\left(\phi(x_j), u_j\right)$ 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:

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

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

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

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:

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

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

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

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

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:

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

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 $\mathbf{Z_{1:P}} \in \mathbb{R}^{N_z \times P} = [z_1 \; z_2 \; ... \; z_P]$ 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 cofficients 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.

4 Results

All continuous dynamics were discretized with an explicit fourth-order Runge Kutta integrator.

4.1 Training

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 in [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.

4.2 Cartpole

As a simple benchmark example, we use the canonical cartpole system. In lieu of an actual hardware experiment (left for future work), we specify two separate analytical models used in simulation: the *nominal* model is a simple cartpole system without friction or damping, whereas the *simulated* model, used exclusively for simulating the system in place of a real hardware platform, includes viscous damping on both degrees of freedom, a tanh Coloumb friction model for the cart, and a deadband on the control signal. Additionally, we altered the mass of the cart and pole from the nominal model by 20% and 25%, respectively. See the provided code for the actual values and models used in the experiments.

We split the analysis of this system into two separate tasks: stabilization about the upward unstable equilibrium from perturbed initial condtions, and the swing-up task where the system must successfully stabilize after starting from the downward equilibrium.

4.2.1 Stabilization

For stabilizing the system about the upward unstable equilibrium, we used an LQR controller designed using to nominal model to collect trajectories on the simulated system. To learn the bilinear models, we used the following nonlinear mapping: $\phi(x) = [\ 1,\ x,\ \sin(x),\ \cos(x),\ \sin(2x)\ ] \in \mathbb{R}^{17}$. After learning both models, an MPC controller was designed using the nominal, eDMD, and jDMD models. For the learned bilinear models, MPC controllers were designed using the "lifted" Jacobians of the bilinear dynamics and the "projected" Jacobians (9). The MPC controllers linearized the dynamics about the equilibrium of $x = [\ 0,\ \pi,\ 0,\ ,0\ ]$ and solved the resulting equality-constrained quadratic program using Riccati recursion and a horizon of 41 time steps (2 seconds).

To analyze sample efficiency of the algorithms, we trained the bilinear models with an increasing number of samples. Each controller was tested using 100 different initial conditions sampled from a uniform distribution of initial conditions centered about the upward equilibirum. The average L2 error of the state after 4 seconds and the upward equilibrium was recorded for each controller. The minimum number of training trajectories to get performance better than the nominal MPC controller is reported in Figure 1. As shown, the proposed approach is more sample efficient even for this relatively simple task than traditional eDMD. It also shows the benefit of applying the control in the original state space by projecting the linearization back to the original states. To our knowledge, no previous works on applying DMD or Koopman operator theory to controlled systems have used this technique of projecting back into the original state space, although the benefits are immediately apparent: with just a few training trajectories (3 for jDMD and 7 for eDMD) we can learn a model that improves upon our nominal LQR controller policy.

4.3 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:

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

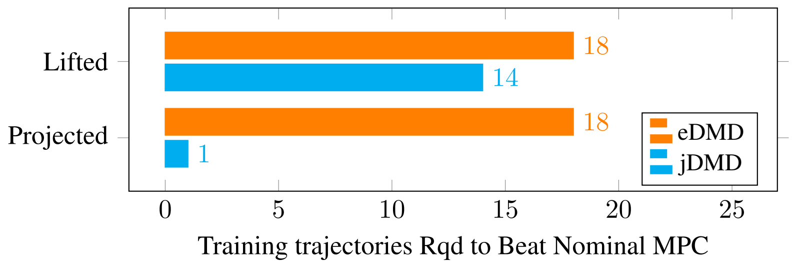


Figure 1: Number of training trajectories requires to beat the nominal MPC controller. The criteria is the L2 norm of the state from the goal state after 4 seconds. The "Lifted" MPC controllers compute the MPC solution in the lifted state space (17 states), whereas the "Projected" MPC contollers project the Jacobians of the bilinear system back into the original state space.

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 assemble an upper triangular factorization of $A^T A$ by forming the factor of $A_1^T A_1$, and updating this factor with each additional batch. To do this, first a matrix "square-root" U is defined as:

$$U = \sqrt{M + W} \quad \rightarrow \quad U^T U = M + W, \tag{20}$$

for two arbitrary matrices M and W. As shown in TODO: cite altro once bib is in, the upper-triangular matrix square root of a sum of matrices can be expressed as a function of each individual matrix square root by using a function QR_R that takes the QR decomposition of the input and returns only the upper triangular R. Using this with the variables introduced in (20) results in the following:

$$\sqrt{M + W} = \text{QR}_\text{R}\left(\begin{bmatrix} \sqrt{M} \\ \sqrt{W} \end{bmatrix}\right) \tag{21}$$

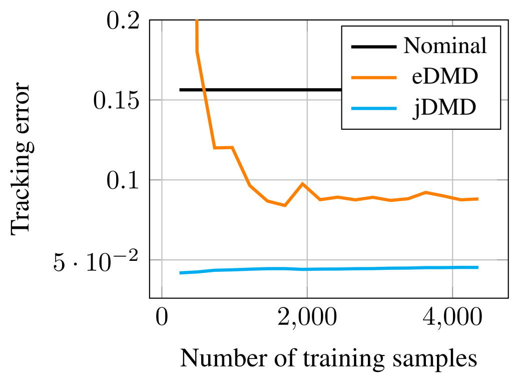
Using this, the factorization of the $F^T F$ matrix can be updated using the following recursion:

$$\sqrt{F_{1:i}^T F_{1:i}} = \text{QR}_\text{R}\left(\begin{bmatrix} \sqrt{F_{1:i-1}^T F_{1:i-1}} \\ F_i \end{bmatrix}\right). \tag{22}$$

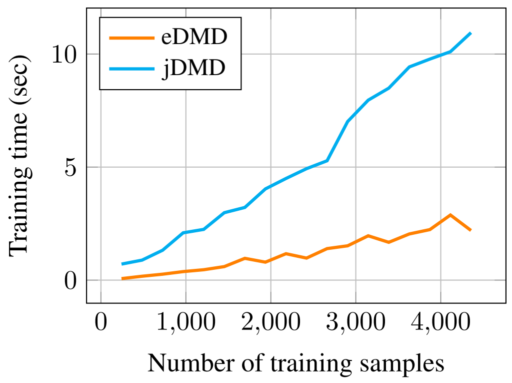
The final algorithm for solving a least squares problem in a recursive-batch fashion is described in algorithm 1. This algorithm can be modified to handle L2 regularized least squares problems by simply replacing line 2 of algorithm 1 with $U \leftarrow \text{QR}_\text{R}(\text{vcat}(F_1, \sqrt{\rho}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

Algorithm 1 Recursive Batch Least Squares with QR

```
1: input $F$, $d$                                  ▷ problem data
2: $U \leftarrow \text{QR}_\text{R}(F_1)$          ▷ 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:     $U \leftarrow \text{QR}_\text{R}(\text{vcat}(U, F_i))$   ▷ 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: ouput $x \leftarrow U^{-1} U^{-T} b$            ▷ forward and backwards substitution to solve for $x$
```



(a) MPC tracking error vs training samples for the cartpole. Tracking error is defined as the average L2 error over all the test trajectories between the reference and simulated trajectories. The proposed jDMD method immediately learns a model good enough for control with just a couple hundred dynamics samples (2 swing-up trajectories).



(b) Training time for cartpole models as a function of training samples. The training time complexity is approximately linear.

4.3.1 MPC Tracking Performance

To train the bilinear model for the swing-up task, we generated 50 training swing-up trajectories using ALTRO, an open-source nonlinear trajectory optimization solver [22, 23], on the nominal cartpole model. Each trajectory was 5 seconds long and sampled at 20 Hz. A linear MPC controller was then used to track the swing-up trajectories on the simulated system, resulting in significant tracking error due to the model mismatch. After learning both models, a linear MPC policy was used to track the original swing-up trajectory generated by ALTRO. The MPC policy linearized the dynamics about the reference trajectory, used a quadratic penalty on deviations from the reference, and was solved using Riccati recursion for a horizon of 41 time steps (2 seconds). To successfully stabilize the system at the upward equilibrium, the MPC policy extrapolated the terminal state and control reference indefinitely. For each of the bilinear models, the Jacobians were projected back onto space of the original dynamics, resulting in an MPC policy that was just as efficient to solve online as the nominal model (all controllers ran at several thousand Hertz without much performance tuning).

The tracking error, defined as the average L2 norm of the error between the reference trajectory and the actual trajectory of the simulated system for 10 test trajectories not included in the training data, was recorded after training both the eDMD and jDMD models with an increasing number of trajectories. The results in Figure 2a clearly show that jDMD produces a high-quality model with extremely few samples, whereas eDMD—even after many training samples—never achieves the same level of performance. The lack of progress with increasing samples is likely a result of poor variety in the training data: after enough samples both methods effectively learn all the information that can be learned from the distribution from which the training data is sampled. This example highlights the value of adding even just a little derivative information to a data-driven approach: it dramatically increases sample efficiency while also improving the quality of the learned model, especially when that model is used in optimization-based controllers such as MPC that rely on derivative information. For reference, the training times are shown in Figure 2b. Note that the training algorithms haven't yet been optimized for max performance but reflect a decent first implementation.

4.4 Planar Quadrotor

As another benchmark example, we use the planar quadrotor model, a simplification of the full quadrotor system with only 3 degrees of freedom. Like the cartpole, we once again specify two separate analytical models used in simulation: the *nominal* model is a simple planar quadrotor system without aerodynamic drag, whereas the "real" *simulated* model includes aerodynamic drag, which can be seen as a sort of viscous damping force. Additionally, we altered the system properties (e.g. mass, rotor arm length, etc.) by 5% to incorporate additional mismatch between the nominal and simulated model. See the provided code for the actual values and models used in the experiments.

Analysis of the planar quadrotor system is also split into 2 separate tasks: stabilization about the hover position with perturbed initial conditions, and a point-to-point translation where the system must do its best to track an infeasible linear trajectory and stabilize about hover at the goal position.

4.4.1 Stabilization

For stabilizing the system about the upward unstable equilibrium, we used an LQR controller designed using the nominal planar quadrotor model to collect various trajectories. For learn our bilinear eDMD and jDMD, the following nonlinear mapping was used: $\phi(x) = [\ 1,\ x,\ \sin(x),\ \cos(x),\ 2x^2 - 1\ ] \in \mathbb{R}^{25}$. After learning both models, respective LQR controllers were designed using the bilinear models' "projected" Jacobians (9). The LQR controllers linearized about hover at the origin ($x = [\ 0,\ 0,\ 0,\ ,0\ 0\ 0\ ]$) and solved the resulting equality-constrained quadratic program using Riccati recursion. Each controller was tested using 30 different initial conditions sampled from a uniform distribution of initial conditions centered about the hover position.

To study how robust eDMD and jDMD are to regularization, we trained multiple bilinear eDMD and jDMD models with increasing L2 (Tikhonov) regularization values. LQR controllers were designed for each model and tested with 100 initial conditions sampled from a uniform distribution centered about the hover position. The average L2 error of the state after 5.0 seconds of simulation, which we'll refer to in this section as the stabilization error, was recorded for each model and respective LQR controller. As shown in Figure 3, the proposed jDMD method remains robust over a range of small regularization values ($< 10^{-1}$) when compared to nominal eDMD. This may suggest a need for greater caution when choosing a regularization value when performing eDMD so that stability and controllability is not lost when attempting to reduce overfitting.

As suggested by Figure 3, eDMD without regularization performs best, but may be prone to overfitting. Therefore, we test eDMD's and jDMD's respective LQR controllers on initial conditions sampled from increasingly larger uniform distributions to analyze the generalizability of the lifted, bilinear models. The LQR controllers are tested on 100 samples generated from the uniform distribution, and the average stabilization error is recorded. As seen in Figure 5b, jDMD's LQR controller is much more robust than eDMD, even when the initial conditions go beyond the scope of the training data. This suggests that adding Jacobian information not only makes eDMD more sample efficient, but can also decrease overfitting.

To further study the robustness of eDMD and jDMD, we tasked the respective LQR controllers for eDMD and jDMD to stabilize under increasingly-varying equilibrium positions. 50 equilibrium positions are sampled from a uniform distribution with increasing bounds, which represent the maximum offset from the origin. The samples are used to test the models' LQR controllers before recording the average stabilization error. The results in Figure 5a show that jDMD is very robust to the change in equilibrium and is able to successfully stabilize about the equilibrium despite the increasing offset. This is expected because stabilization performance should be invariant with respect to the quadrotor's equilibrium position, and this information is provided by the Jacobians in jDMD. eDMD's increasing inability to stabilize for greater equilibrium offsets is likely a result of eDMD being overfitted to the training trajectories, which all stabilize about the origin. Therefore, this example once again highlights the value of adding Jacobian information: it is not only a data augmentation technique for sample efficiency, but also a regularizer that can decrease overfitting, making the learned models (and subsequent controllers) more robust and generalizable.

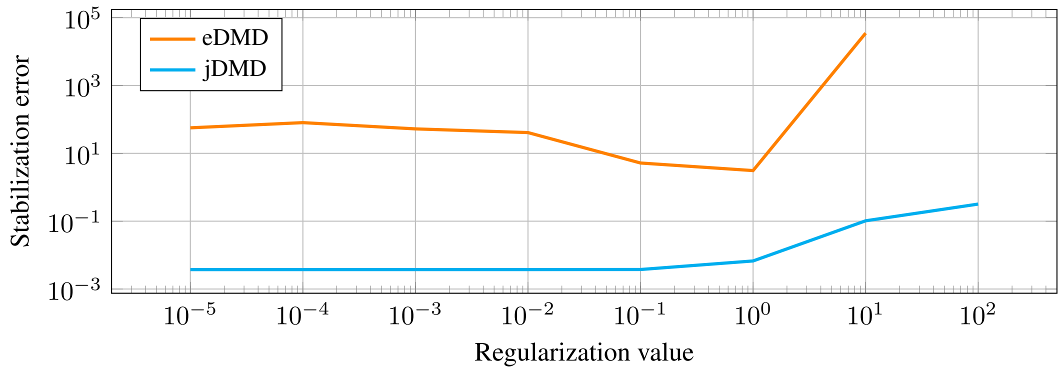
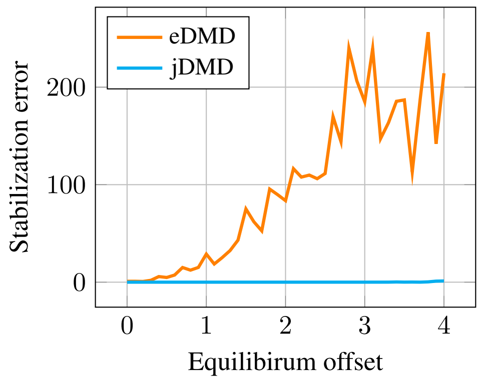
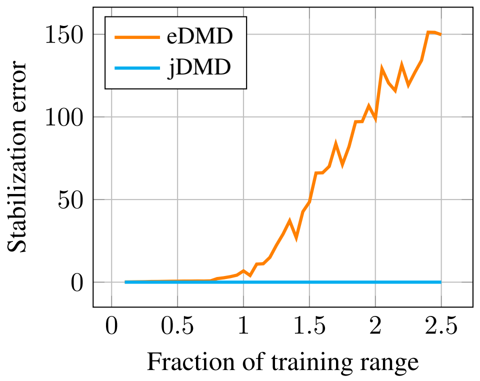


Figure 3: LQR stabilization with equilibrium offset

Figure 4: LQR stabilization error over a range of L2 regularization values. jDMD is much more robust and consistent over small regularization values.



(a) LQR stabilization error over increasing equilibrium offset



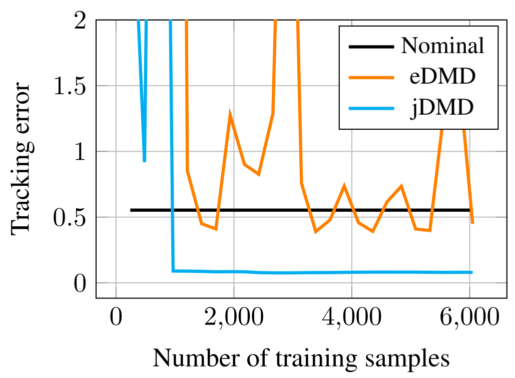
(b) LQR stabilization error over varying range of distribution for sampling initial conditions. A training range fraction greater than 1 indicates the distribution range is beyond that used to generate the training trajectories

Figure 5: LQR controller robustness to varying initial conditions and equilibrium offset. jDMD is much more robust overall with an ability to stabilize outside the scope of training data

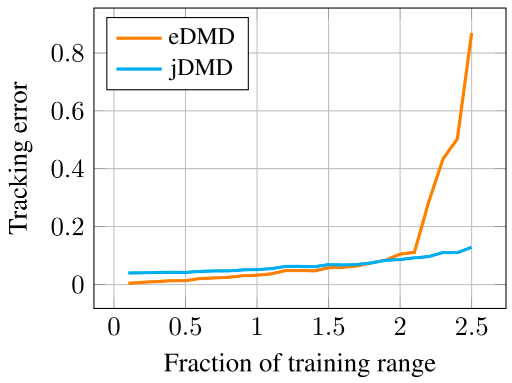
4.4.2 MPC Tracking Performance

To train the bilinear model for the MPC tracking task, we generated 50 infeasible point-point linear trajectories that all ended at the origin with the planar quadrotor in hover. The starting initial condition of the system was sampled from a uniform distribution about the origin. A linear MPC controller was then used to track the infeasible linear trajectories on the simulated system, resulting in significant tracking error due to the model mismatch. After learning both models, a linear MPC policy (identical to the cartpole example) was used to track the original linear trajectory as best as possible.

The tracking error we recorded for 35 test trajectories after training both the eDMD and jDMD models with an increasing number of trajectories. Like the cartpole example, results in Figure 6a clearly show that jDMD produces a high-quality model for MPC trackign with extremely few samples, whereas eDMD —even after many training samples—is never able to develop a model with consistent and similar performance. The lack of progress with increasing samples is likely a result of poor variety in the training data: the jDMD model is not receiving any new information from additional training trajectories, and eDMD is not able to capture enough data that can capture enough



(a) MPC tracking error vs training samples for the planar quadrotor. The proposed jDMD method again learns a model good enough with 1000 samples while eDMD is never able to converge to a good-enough model.



(b) MPC tracking error of quadrotor over varying range of distribution for sampling initial conditions. Drastically increasing error for eDMD suggests greater overfitting.

behaviors that lie in the testing space. This is once again highlighted in the results of Figure 6b, which show that eDMD is only able to successfully track trajectories before falling apart when the range of test trajectories exceeds that of the training data. Meanwhile, jDMD is able to consistently track the trajectory to the goal state, despite having greater tracking error in a certain range of trajectories—this further hints at eDMD overfitting on the training data.

5 Limitations

6 Conclusion

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. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2589–2594, sep 2014. ISSN 10504729. 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. *IEEE International Conference on Intelligent Robots and Systems*, pages 8388–8395, 2021. ISSN 21530866. 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. *IEEE Transactions on Intelligent Vehicles*, 4(2):309–320, jun 2019. ISSN 23798858. 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. *Proceedings of Machine Learning Research*, pages 1–14, 2020. 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. *Proceedings of the Conference on Robot Learning*, 100:990–1004, 2020. ISSN

2640-3498. 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. *Proceedings - IEEE International Conference on Robotics and Automation*, 2021-May:2811–2817, 2021. ISSN 10504729. 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. jan 2022. 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. *IEEE Robotics and Automation Letters*, 6(3):4369–4376, 2021. ISSN 23773766. doi:10.1109/LRA.2021.3068117.

[10] M. Korda and I. Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018. 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. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 9216–9222, may 2020. ISSN 10504729. 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. *Springer Proceedings in Advanced Robotics*, 17:347–363, feb 2020. ISSN 25111264. doi:10.48550/arxiv.2002.09093. URL https://arxiv.org/abs/2002.09093v3.

[13] U. Fasel, E. Kaiser, J. N. Kutz, B. W. Brunton, and S. L. Brunton. SINDy with Control: A Tutorial, 2021. ISSN 25762370. 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. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018. ISSN 15360040. 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. *Journal of Nonlinear Science*, 25(6):1307–1346, dec 2015. ISSN 14321467. 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. *Springer Proceedings in Advanced Robotics*, 17:194–209, 2021. ISSN 25111264. 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. *Proceedings - IEEE International Conference on Robotics and Automation*, 2021-May:3546–3552, 2021. ISSN 10504729. 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. 2022.

[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”. 2006.

[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., 2021. ISBN 9781728190778. doi: 10.1109/ICRA48506.2021.9562002.

[22] T. A. Howell, B. E. Jackson, and Z. Manchester. ALTRO: A Fast Solver for Constrained Trajectory Optimization. *IEEE International Conference on Intelligent Robots and Systems*, pages 7674–7679, nov 2019. ISSN 21530866. doi:10.1109/IROS40897.2019.8967788.

[23] 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. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021. doi:10.1109/ICRA48506.2021.9561438. URL `https://github.com/`.