2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Episodic Koopman Learning of Nonlinear Robot Dynamics with Application to Fast Multirotor Landing

Carl Folkestad*, Daniel Pastor*, and Joel W. Burdick

***Abstract*— This paper presents a novel episodic method to learn a robot's nonlinear dynamics model and an increasingly optimal control sequence for a set of tasks. The method is based on the *Koopman operator* approach to nonlinear dynamical systems analysis, which models the flow of *observables* in a function space, rather than a flow in a state space. Practically, this method estimates a nonlinear diffeomorphism that lifts the dynamics to a higher dimensional space where they are linear. Efficient Model Predictive Control methods can then be applied to the lifted model. This approach allows for real time implementation in on-board hardware, with rigorous incorporation of both input and state constraints during learning. We demonstrate the method in a real-time implementation of fast multirotor landing, where the nonlinear ground effect is learned and used to improve landing speed and quality.**

I. Introduction

While modeling and identification (ID) techniques are well developed for some robotic mechanisms, such as manipulators, there are an increasing number of applications where modeling and ID are difficult. Consider a multirotor drone, whose basic flight mechanics in open air are well understood [1], [2]. However, when multirotors fly close to the ground or a wall, or inside a narrow tunnel (e.g., for non-invasive inspection), the unmodeled effects of the complex vehicle-air-environment interaction can substantially reduce the drone's path tracking accuracy, and perhaps its stability. While ground effect models can be incorporated, their accuracy is limited, and their parameters must still be estimated in a slow process. This particular problem motivates this paper. There are many other applications, such as soft robotic structures or robotic manipulation of soft materials, where first principles modeling and parameter identification remain challenging in practice. Moreover, one must still design a controller for the nonlinear mechanics that are identified.

Learning can capture the salient aspects of a robot's complex mechanics and environmental interactions. Gaussian process dynamical systems models [3] can identify nonlinear affine control models in a non-parametric way. Yet, effective nonlinear control design after identifying the model can be challenging. Model-free reinforcement learning (MFRL) [4] learns feedback policies that implicitly incorporate the robot's dynamics. However, sample efficiency is very low. Moreover, while safety during MFRL is now possible [5], [6], one cannot yet guarantee that learned policies will satisfy performance requirements or state and actuator limits.

*Both authors contributed equally
All authors are with the Division of Engineering and Applied Sciences, California Institute of Technology, Pasadena, CA 91125, USA {carl.folkestad,dpastorm,jburdick}@caltech.edu

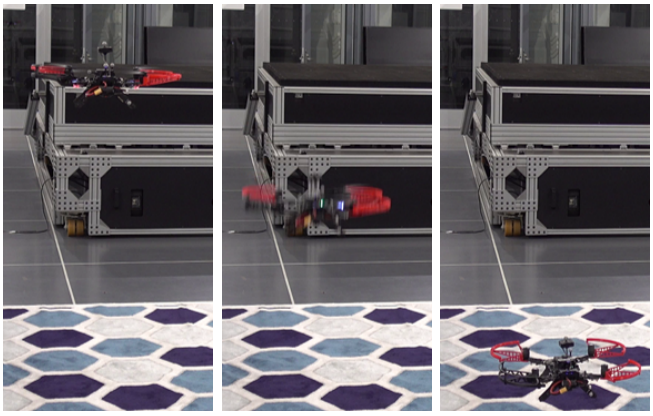

Fig. 1: From left to right: hovering before the sequence start, high speed descent with learned dynamics, and soft landing.

This paper presents a new method, based on *Koopman spectral* analysis, to learn nonlinear robot dynamical models. Conventionally, a system's behavior is characterized via its state space flows. In contrast, Koopman-based approaches study the evolution of *observables*, which are functions over the state-space. In this space, the system can be represented by a *linear* (but possibly infinite dimensional) operator [7], [8]. Practically, the nonlinear dynamics are *lifted* to a higher dimensional space where they are linear. Efficient linear and optimal control principles can be applied to the lifted system.

Koopman inspired modelling and identification techniques have received substantial recent attention [9]. The Dynamic Mode Decomposition (DMD) and extended DMD (EDMD) methods [10], [11] have been successfully used in the field of fluid mechanics to capture low-dimensional structure in complex flows. More recently, Koopman-style modeling has been extended to *controlled* nonlinear systems [12], [13].

In practice, to model and identify a nonlinear system in the Koopman framework, one can identify a finite dimensional approximation to the linear Koopman operator, or identify the Koopman *eigenfunctions*. We use the latter approach (see Section II-B). Previous methods for identifying Koopman eigenfunctions (e.g., [14], [15]) depend upon assumptions that are problematical for robotic systems: the ID data is gathered while the robot operates under open loop controls, which can lead to catastrophic system damage.

In very recent work [16], the authors and collaborators have developed a *Koopman Eigenfunction Extended Dynamic Mode Decomposition* (KEEDMD) method to identify/learn an unknown nonlinear dynamical system from a batch of data gathered while the (robotic) system operates under a stabilizing (but not necessarily optimal) *linear controller*.

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

This paper substantially extends our prior work [16] to make it practically useful for robotics. First, the method gathers data while the system operates under any *nonlinear* stabilizing controller. This enables input vector field nonlinearities to be captured, unlike prior Koopman-based model ID approaches. Second, we introduce an episodic learning procedure, by considering the closed-loop dynamics obtained with a non-linear controller as the autonomous dynamics for the next episode. This feature increases sample efficiency (i.e., fewer learning trials) for improving specific tasks, and enables nonlinear actuation effects, which are important in robotics, to be captured in the Koopman eigenfunctions. Third, it should be noted that data collected from robots executing trajectories formally violates the i.i.d. assumption underlying the performance guarantees of most learning paradigms. In practice, this fact can lead to error cascades and poor performance guarantees. Episodic learning mitigates this problem [17]. Finally, our method integrates Model Predictive Control (MPC) [18] into its structure, thereby allowing control and state constraints to be satisfied during the learning process.

Previous approaches have implemented MPC in real time on modest computational hardware [19], on multirotors using an explicit solution in simulation [20], and designed feedback linearizing controllers for multirotors in real experiments [21], [22]. Bouffard *et al*. [23] also used MPC to learn ground effects using an experimental multirotor, but used an Extended Kalman Filter (EKF) in combination with Learning-Based MPC (LBMPC). Shi *et al*.[24] experimentally demonstrated using a spectrally-normalized neural network to learn the ground effect and improve drone landing by designing a feedback linearizing controller utilizing the learned model. We introduce a new approach to solve this problem and aim to demonstrate that our method represents a first step towards practical Koopman-based learning and control of real-world robotic systems.

Section II reviews relevant facts about the Koopman operator and KEEDMD [16]. Sections III and IV introduce and develop the first main paper contribution–the episodic KEEDMD algorithm. Section V presents our second contribution, the experimental demonstration that our method can learn the ground effect in a fast landing multirotor.

II. Preliminaries on the Koopman Eigenfunction Extended Dynamic Mode Decomposition

A. Koopman Operator

Consider the autonomous dynamical system:

$$\dot{\mathbf{x}} = f(\mathbf{x}) = A\mathbf{x} + v(\mathbf{x}) \tag{1}$$

with the state $x \in \mathcal{X} \subset \mathbb{R}^n$ and f Lipschitz continuous on $\mathcal{X}$. We assume that the system (1) has a fixed point at the origin, $f(0) = 0$. The flow of this dynamical system is denoted by $S_t(x)$ and is defined as

$$\frac{d}{dt} S_t(x) = f(S_t(x)) \tag{2}$$

for all $x \in \mathcal{X}$ and all $t \geq 0$. The *Koopman operator semigroup* $(U_t)_{t\geq 0}$, from now on simply denoted as the *Koopman operator*, is defined as

$$U_t g = g \circ S_t \tag{3}$$

for all $g \in \mathcal{C}(\mathcal{X})$, where $\circ$ denotes the function composition and $\mathcal{C}(\mathcal{X})$ is the space of all continuous functions on $\mathcal{X}$. Each element of the Koopman operator maps continuous functions to continuous functions, $U_t : \mathcal{C}(\mathcal{X}) \to \mathcal{C}(\mathcal{X})$. Crucially, each U_t is a *linear* operator. An *eigenfunction* of the Koopman operator associated to an eigenvalue $e^{\lambda} \in \mathbb{C}$ is any function $\phi \in \mathcal{C}(\mathcal{X})$ that defines a coordinate evolving linearly along the flow of (1) satisfying

$$(U_t\phi)(x) = \phi(S_t(x)) = e^{\lambda t}\phi(x) \ . \tag{4}$$

B. Data-driven Construction of Koopman Eigenfunctions

For any sufficiently smooth autonomous dynamical system that is asymptotically stable to a fixed point, Koopman eigenfunctions can be constructed by finding the eigenfunctions of the system's linearization around the fixed point and then composing them with a diffeomorphism [14]. The linearization of the dynamics (1) around the origin is

$$\dot{\mathbf{y}} = \mathbf{D}f(0)\mathbf{y} = \hat{A}\mathbf{y}, \ \mathbf{y} \in \mathcal{Y} \tag{5}$$

The following proposition describes how to construct eigenfunction-eigenvalue pairs for the linearized system (5).

Proposition 1. *Let $\hat{A}$ denote the linearization (5) of nonlinear system (1) and let $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ be a basis of the eigenvectors of $\hat{A}$ corresponding to nonzero eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$. Let $\{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ be an adjoint basis to $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ such that $\langle \mathbf{v}_q, \mathbf{w}_r \rangle = \delta_{qr}$ and $\mathbf{w}_q$ is an eigenvector of $\hat{A}^*$ at eigenvalue $\bar{\lambda}_q$. Then, the linear functional*

$$\psi_q(\mathbf{y}) = \langle \mathbf{y}, \mathbf{w}_q \rangle \tag{6}$$

is a nonzero eigenfunction of $U_{\hat{A}}$, the Koopman operator associated to $\hat{A}$. Furthermore, for any $(m_1, \ldots, m_d) \in \mathbb{N}_0^d$

$$\left(\prod_{q=1}^{d} e^{m_q \lambda_q}, \prod_{q=1}^{d} \psi_q^{m_q} \right) \tag{7}$$

is an eigenpair of the Koopman operator $U_{\hat{A}}$.

These linear functionals (6), termed *principal eigenfunctions*, are used to construct the eigenfunctions associated with the Koopman operator of the nonlinear dynamics through the use of a *conjugacy map* (See [9], Prop. 7).

Proposition 2. *Assume that the nonlinear system (1) is topologically conjugate to the linearized system (5) via the diffeomorphism $h : \mathcal{X} \to \mathcal{Y}$. Let $B \in \mathcal{X}$ be a simply connected, bounded, positively invariant open set in $\mathcal{X}$ such that $h(B) \subset Q_r \subset \mathcal{Y}$, where Q_r is a cube in $\mathcal{Y}$. Scaling Q_r to the unit cube Q_1 via the smooth diffeomorphism $g : Q_r \to Q_1$ gives $(g \circ h)(B) \subset Q_1$. Then, if ψ is an eigenfunction for $U_{\hat{A}}$ at e^{λ}, then $\psi \circ g \circ h$ is an eigenfunction for U_f at eigenvalue e^{λ}, where U_f is the Koopman operator associated with the nonlinear dynamics (1).*

An extension of the Hartman-Grobman theorem ([7], Theorem 2.3) guarantees the existence of a $\mathcal{C}^1$ diffeomorphism

$$y = c(\mathbf{x}) = \mathbf{x} + h(\mathbf{x}) \tag{8}$$

between the linearized and nonlinear systems in the entire basin of attraction of a fixed point, such that $\mathbf{D}c(\mathbf{0}) = I$.

C. KEEDMD with Trajectory-tracking Control Laws

Data-driven construction of Koopman eigenfunctions for nonlinear dynamics of the form $\dot{\mathbf{x}} = a(\mathbf{x}) + B\mathbf{u}$ is based on Section II-B, and summarized in Algorithm 1. General nonlinear dynamics will be considered in Section III. For trajectory-tracking control laws, the algorithm assumes that a linearized nominal dynamics model $\dot{\mathbf{y}} = A_{nom}\mathbf{y} + B_{nom}\mathbf{u}$ is known, along with a nominal stabilizing feedback control law $\mathbf{u}(t) = K_{nom}(\mathbf{y}(t) - \boldsymbol{\tau}(t))$, where $\boldsymbol{\tau}(t) \in \mathcal{X}$ is the trajectory we want the system to track. Typically, A_{nom}, B_{nom} is the linearization of the dynamics around the origin and K_{nom} the controller gains determined by e.g. LQR on the nominal state space model. Principal eigenfunctions, with eigenvalues e_j^λ, of the Koopman operator for the closed loop linearized dynamics $\dot{\mathbf{y}} = (A_{nom} + B_{nom}K_{nom})\mathbf{y}$ are constructed using Prop. 1: $\psi_j(\mathbf{y}) = \langle \mathbf{y}, \mathbf{w} \rangle$, where $\mathbf{w}$ is an adjoint basis of the eigenvectors of $A_{cl} = (A_{nom} + B_{nom}K_{nom})$. Products and powers (7) generate arbitrarily many eigenpairs of the linearized system before applying the diffeomorphism (8) between the nonlinear and linearized dynamics. This diffeomorphism $h : \mathcal{X} \to \mathcal{Y}$ is learned in a supervised way (*e.g.* a neural network trained with gradient descent) by performing empirical risk minimization (ERM) of an appropriate loss function over a model class $\mathcal{H}_h$. For the trajectory-tracking case, the loss function is of the form

$$\begin{aligned} &\mathcal{L}_h(\mathbf{x}, \dot{\mathbf{x}}, A_{cl}\mathbf{x} - \dot{\mathbf{x}}, \boldsymbol{\tau}(t)) = \\ &\quad ||\dot{h}(\mathbf{x}) - A_{cl}h(\mathbf{x}) - (A_{cl}\mathbf{x} - \dot{\mathbf{x}}) + B_{nom}K_{nom}\boldsymbol{\tau}||^2 \end{aligned} \tag{9}$$

(see [16] for details). Finally, a function scaling $\mathcal{Y} \subset \mathcal{Q}_r$ into Q_1, where $\mathcal{Q}_r$ is a hyper cube of the same dimension as $\mathcal{Y}$ with radius r, is constructed (i.e. by scaling each coordinate into a unit cube) and approximate Koopman eigenpairs for the unknown, nonlinear dynamics are constructed: $(e^{\lambda_i}, \phi_i) = (\tilde{e}_i^\lambda, \tilde{\psi}_i(g(h(\mathbf{y}))))$, where $\tilde{e}_i^\lambda, \tilde{\psi}_i$ are the eigenpairs constructed with (7).

Algorithm 1 Data-driven Koopman Eigenpair Construction

Require: Data set $\mathcal{D} = \left((\mathbf{x}_k^j, \mathbf{u}_k^j)_{k=0}^{M_s}\right)_{j=1}^{M_t}$, nominal model matrices A_{nom}, B_{nom}, nominal control gains K_{nom}, desired trajectory $\boldsymbol{\tau}(t)$, number of lifting functions N, N power combinations $(m_1^{(i)}, \ldots, m_d^{(i)}) \in \mathbb{N}_0^d, i = 1, \ldots, N$

Construct principal eigenpairs for the linearized dynamics: $(e_j^\lambda, \psi_j(\mathbf{y})) \leftarrow (e_j^\lambda, \langle \mathbf{y}, \mathbf{w}_j \rangle), \quad j = 1, \ldots, n$
Construct N eigenpairs from the principal eigenpairs: $(\tilde{e}_i^\lambda, \tilde{\psi}_i) \leftarrow \left(\prod_{j=1}^d e^{\lambda_j^{m_j^{(i)}}}, \prod_{j=1}^d \psi_j^{m_j^{(i)}}\right), \quad i = 1, \ldots, N$
Fit diffeomorphism estimator: $h(\mathbf{y}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D})$
Construct scaling function: $g(\mathbf{y}) \leftarrow g : \mathcal{Q}_r \to \mathcal{Q}_1$
Construct N eigenpairs for the nonlinear dynamics: $(\tilde{e}_i^\lambda, \phi_i) \leftarrow (\tilde{e}_i^\lambda, \tilde{\psi}_i(g(h(\mathbf{y})))), \quad i = 1, \ldots, N$

Output: $\Lambda = \text{diag}(\tilde{e}_1^\lambda, \ldots, \tilde{e}_N^\lambda), \quad \boldsymbol{\phi} = [\phi_1, \ldots, \phi_N]^T$

KEEDMD uses the constructed eigenfunctions to lift the system states to a higher dimensional space where a linear dynamical model of the form $\dot{\mathbf{z}} = A\mathbf{z} + B\mathbf{u}$ can be identified. For Lagrangian dynamics, as the example in Section V, we use $\mathbf{z} = [\mathbf{x}, \boldsymbol{\phi}(\mathbf{x})]^T$ as the lifted state, where $\mathbf{x} = [\mathbf{p} \quad \mathbf{v}]^T$, $\mathbf{p}$ the position, $\dot{\boldsymbol{p}} = \mathbf{v}$ the velocity, and $\boldsymbol{\phi}$ is a vector of the eigenfunctions. While helping data efficiency, the method does not generally require any *a priori* information of the structure of the dynamics. Since the time evolution of the eigenfunctions is dictated by their eigenvalues, we can show that the lifted state space model has the structure

$$\begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{v}} \\ \dot{\boldsymbol{\phi}}\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}\right) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I & 0 \\ A_{\mathbf{vp}} & A_{\mathbf{vv}} & A_{\mathbf{v}\boldsymbol{\phi}} \\ -B_{\boldsymbol{\phi}}K_{nom} & \Lambda \end{bmatrix}}_{A} \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \boldsymbol{\phi}\left(\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}\right) \end{bmatrix} + \underbrace{\begin{bmatrix} B_{\mathbf{p}} \\ B_{\mathbf{v}} \\ B_{\boldsymbol{\phi}} \end{bmatrix}}_{B} \mathbf{u}$$

where $0, I, \Lambda, K_{nom}$ are fixed matrices and $A_{\mathbf{vp}}, A_{\mathbf{vv}}$, $A_{\mathbf{v}\boldsymbol{\phi}}$, $B_{\mathbf{p}}$, $B_{\mathbf{v}}$, $B_{\boldsymbol{\phi}}$ are determined from data, and the term $-B_{\boldsymbol{\phi}}K_{nom}$ accounts for the nominal controller's state feedback effect on the evolution of the eigenfunctions. The desired trajectory effect of the controller is captured by the learning framework. The unknown elements of A and B can be found using linear regression on the data collected under the nominal control law (see [16] for details). We define $\mathcal{L}_z$ as the mean squared error for the regression problem, where different forms of regularization can be included if needed. As the feedback controller is state-dependent, it is not possible to disambiguate its effect from the passive uncontrolled system dynamics in the learning process. To avoid an ill-conditioned KEEDMD regression problem, Brownian noise is added to perturb the nominal controller [25]. Brownian noise is chosen in this instance because pure sampling from e.g. a Gaussian distribution leads to perturbations that have too high frequency to perturb the movement of the multirotor. This perturbation is also used by our episodic learning framework (Section IV). When the lifted state space model is identified, state estimates can be obtained as $\mathbf{x} = C\mathbf{z}$, where $C = [I \quad 0]$. C is denoted the *projection matrix* of the lifted state space model.

III. Episodic KEEDMD learning

A. Problem Setup and Dynamics Modeling

Assume that we have selected a fixed trajectory $\boldsymbol{\tau}$ to be tracked by the robot during episodic learning. Further assume a nominal controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$ that can stabilize the system to $\boldsymbol{\tau}$ within a region of attraction Ω around the trajectory. This controller might be the outcome of a previous learning episode (see below), or the simple linear nominal controller from KEEDMD (Section II). Finally, the system's governing dynamics are assumed to be *unknown*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{10}$$

where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n, u \in \mathcal{U} \subset \mathbb{R}^m$. and $f(\mathbf{x}, \mathbf{u})$ is assumed to be Lipschitz continuous on $\mathcal{X} \times \mathcal{U}$.

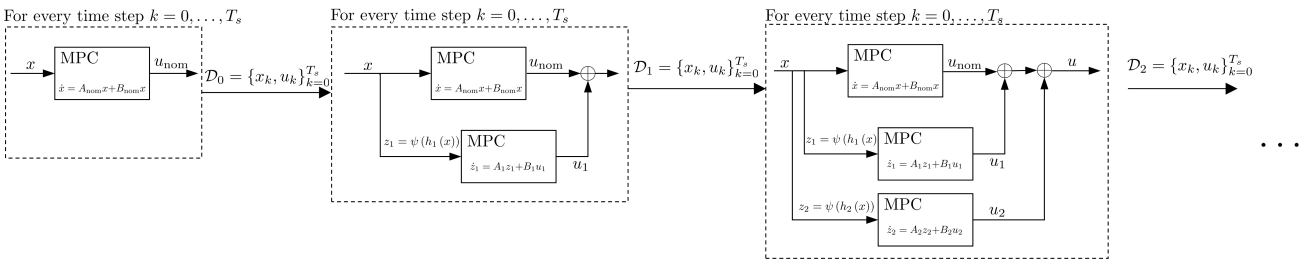

Fig. 2: Flow chart showing the different elements for each episode.

B. Learning with Arbitrary Stabilizing Control Laws

KEEDMD (Section II) requires batch training data to be collected from system executions under a nominal linear control law: $\mathbf{u}_{nom}(\mathbf{x}) = K_{nom}(\mathbf{x} - \boldsymbol{\tau}(t))$. A main contribution of this paper is to iteratively learn an improving sequence of eigenfunctions and nonlinear controllers. Specifically, we will iteratively use the lifted state-space model to design an MPC-controller to track learning trajectories (see Figure 2).

If a candidate nonlinear controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$ can stabilize system (10) to a given trajectory $\boldsymbol{\tau}$, the controlled system can be described by the autonomous dynamics

$$\dot{\mathbf{x}} = f(\mathbf{x}, \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)) = F_{\hat{\mathbf{u}}, \boldsymbol{\tau}}(\mathbf{x}, t). \tag{11}$$

Importantly, for the autonomous dynamics (11), there exists an associated Koopman operator $U_{F_{\hat{\mathbf{u}}, \boldsymbol{\tau}}}$ that depends on control law $\hat{\mathbf{u}}$ and trajectory $\boldsymbol{\tau}$. Therefore, approximate eigenpairs for $U_{F_{\hat{\mathbf{u}}, \boldsymbol{\tau}}}$ can be constructed (see Section II-B) from the gathered state and control samples. A lifted state-space model can be constructed from these eigenpairs.

However, unlike the framework reviewed in Section II, we aim to learn a dynamical model that assumes that the system is already regulated by the *nominal controller* $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$. As a result, the A-matrix of the lifted state space model captures the autonomous dynamics under the nominal control law (Eq. 11), and the B-matrix captures the effect of control variations around the nominal controller:

$$\dot{\hat{\mathbf{z}}} = A\hat{\mathbf{z}} + B(\mathbf{u}(\mathbf{x}, \boldsymbol{\tau}, t) - \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)). \tag{12}$$

This model is used in an MPC framework below to design an *augmenting* control law that adds optimal control actions to the nominal controller. The augmenting controller leverages the improved system model to make corrections to sub-optimal actions taken by the nominal controller.

C. Modifications to Allow the Diffeomorphism to Capture Nonlinear Control and Dynamics Effects

To enable the learning framework to capture nonlinear effects caused by the nonlinear controller and actuated dynamics, a minor modification to the function approximator of h is necessary. Namely, since the diffeomorphism is affected by the forcing signal $\boldsymbol{\tau}(t)$ it must be included in the inputs of h. This is motivated by the form of the diffeomorphism loss function (9). In the case considered in the preliminaries however, the actuated dynamics and controller are assumed to be linear. This causes the effect of the forcing signal $\boldsymbol{\tau}(t)$ to cancel out such that the diffeomorphism is independent of the desired trajectory. In the general nonlinear case however, the effect is not canceled out and must be captured by the diffeomorphism. As a result, the diffeomorphism is modified such that $h : \mathcal{X} \times \mathcal{X} \to \mathcal{Y}$ (see [16] for details).

IV. Episodic Eigenfunction Construction and KEEDMD Inference

This section describes the main contribution of this paper, a substantial extension of the KEEDMD framework to allow iterative learning and improvement of the lifted state-space model and its associated controller.

A. Overview of the Episodic Learning Algorithm

Algorithm 2 summarizes our episodic learning approach, which applies three key steps per episode. In each episode, e, the first key step starts when an initial condition is sampled from set X_0 and an experiment is executed with the controller that results from the previous episode $\mathbf{u}_{e-1}(\mathbf{x}, \boldsymbol{\tau}, t)$. The state $\mathbf{x}$, control actions $\mathbf{u}_{e-1}$, Brownian noise control perturbations $\tilde{\mathbf{u}}$, and the desired position dictated by the trajectory at the time associated with the i-th sample $\boldsymbol{\tau}_i$ are sampled. State data can be differentiated numerically to find estimates $\dot{\mathbf{x}}$. The resulting data set is:

$$\mathcal{D}_x^{(e)} = \left\{ \left(\mathbf{x}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{x}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{13}$$

where $\mathbf{x}_i^{(e)}$ denotes the i-th timestep of the e-th episode and T_s denotes the number of samples in the episode. From $\mathcal{D}_x^{(e)}$ we estimate the diffeomorphism h and construct the eigenfunctions $\boldsymbol{\phi}^{(e)}(\mathbf{x})$ with associated eigenvalues $\Lambda^{(e)}$, via Algorithm 1. Since changes in the control law between episodes are expected to be small, we warm start the learning algorithm with model coefficients from the previous episode.

The second key step is to use the constructed eigenpairs to build a lifted data set $\mathcal{D}_z^{(e)}$.

$$\mathcal{D}_z^{(e)} = \left\{ \left(\mathbf{z}_i^{(e)}, \mathbf{u}_i^{(e)}, \tilde{\mathbf{u}}_i^{(e)}, \boldsymbol{\tau}_i \right), \dot{\mathbf{z}}_i^{(e)} \right\}_{i=1}^{T_s} \tag{14}$$

which is the same data as $\mathcal{D}_x^{(e)}$, but with the state and its derivative, $\mathbf{x}_i^{(e)}, \dot{\mathbf{x}}_i^{(e)}$, replaced with the lifted state and its derivative, $\mathbf{z}_i^{(e)}, \dot{\mathbf{z}}_i^{(e)}$. Next, data from the current and previous episodes is aggregated: $\bigcup_{j=1}^{e} \mathcal{D}_z^{(j)}$. The lifted state-space

model is constructed from this data using the framework of Section III-B. This results in a model of the form (12).

In the third and final step, an augmenting MPC is designed (see Section IV-B) for the lifted state-space model. The evaluation of the previous iteration's controllers is necessitated by the fact that the eigenfunctions depend on the dynamics under closed loop control with the controller deployed in the previous episodes. The controller augmentations are weighted and added to the previous episode's control law: $\mathbf{u}_e = \mathbf{u}_0 + \sum_{j=1}^{e} w_j \mathbf{u}_j$, where w_e is a weighting factor indicating the confidence in the augmenting controller. The weighting factors can be any monotonically increasing sequence on the interval $[0, 1]$ which allows the augmenting controller to have a bigger impact after a sufficiently rich data set has been collected.

B. Model Predictive Controller Details

Inspired by [26], we transform the original non-linear optimization problem into an efficient quadratic program (QP). The QP formulation requires us to discretize the previously learned linear continuous dynamics. We assume a known objective function of states and controls only. For simplicity, we use a quadratic objective function with respect to the state error and control action, but other objective functions can be used by simply adding it to the lifting functions. We assume known control bounds $u_{\min}, u_{\max} \in \mathbb{R}^m$ and state bounds $x_{\min}, x_{\max} \in \mathbb{R}^n$. Because the control input for each MPC problem refers to the change from the the previous controller, we have to correct for this change in the control bounds. All these assumptions define the following optimization problem that is solved at each time step:

$$\begin{aligned} \min_{\substack{u\in\mathbb{R}^{m\times N_p}\\ z\in\mathbb{R}^{N\times N_p}}} \ & \sum_{p=1}^{N_p} \left[(C_j z_p - \tau_p)^T Q (C_j z_p - \tau_p) + u_p^T R u_p \right] \\ \text{s.t.} \quad & z_p = A_j z_{p-1} + B_j u_p \\ & x_{\min} \le C_j z_p \le x_{\max} \qquad p = 1, \dots, N_p \\ & u_{\min} \le u_p - \textstyle\sum_{i=1}^{j-1} w^{(i)} u_p^{(i)} \le u_{\max} \\ & z_0 = \phi_j(x_k) \end{aligned} \tag{15}$$

Algorithm 2 Episodic KEEDMD

Require: Desired trajectory $\boldsymbol{\tau}$, nominal controller $\hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$, diffeomorphism model class $\mathcal{H}_h$, diffeomorphism loss $\mathcal{L}_h$, number of lifting functions N, KEEDMD loss $\mathcal{L}_z$

$\mathcal{D}_z = \emptyset, \quad \mathbf{u}_0(\mathbf{x}, \boldsymbol{\tau}, t) = \hat{\mathbf{u}}(\mathbf{x}, \boldsymbol{\tau}, t)$

for $e = 1, \dots, N_{ep}$ **do**
- Sample initial condition: $\mathbf{x_0} \leftarrow \text{sample}(X_0)$
- Execute experiment: $\mathcal{D}_x^{(e)} \leftarrow \text{run}(\mathbf{x}_0, \mathbf{u}^{(e\ \ 1)}(\mathbf{x}, \boldsymbol{\tau}, t))$
- Fit diffeomorphism estimator: $h(\mathbf{x}) \leftarrow \text{ERM}(\mathcal{H}_h, \mathcal{L}_h, \mathcal{D}_x^{(e)})$
- Construct eigenpairs: $(\boldsymbol{\phi}^{(e)}(\mathbf{x}), \Lambda^{(e)}) \leftarrow h(g(\boldsymbol{\psi}(\mathbf{x})))$
- Construct and aggregate lifted data set: $\mathcal{D}_z \leftarrow \mathcal{D}_z \cup \mathcal{D}_z^{(e)}$
- Fit KEEDMD model: $\dot{\mathbf{z}}^{(e)}(\mathbf{z}) \leftarrow \text{ERM}((\boldsymbol{\phi}^{(e)}, \Lambda^{(e)}), \mathcal{L}_z, \mathcal{D}_z)$
- Update controller: $\mathbf{u}^{(e)} \leftarrow \mathbf{u}^{(e\ \ 1)} + w^{(e)} \text{MPC}(\dot{\mathbf{z}}^{(e)}, \mathbf{u}^{(e\ \ 1)})$

end for

Output: Final control law $\mathbf{u}^{(N_{ep})}$

where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive semidefinite cost matrices, $\tau \in \mathbb{R}^{n\times N_p}$ is the reference trajectory, $A_j \in \mathbb{R}^{N\times N}$ and $B_j \in \mathbb{R}^{N\times m}$ are the discrete time versions of (12) for controller j, $C_j \in \mathbb{R}^{n\times N}$ is the j^{th} controller's projection matrix, and $\phi_j \in \mathbb{R}^N$ are the j^{th} controller's eigenfunctions. See Figure 2 to see how each controller is used as more episodes are being executed. In addition, we add a smoothing regularizer to avoid chatter that may arise from optimization-based controllers [27] of the form $\sum_{p=1}^{N_p} \alpha_R (u_p - u_{p-1})^2$ where u_0 is the deployed control action at the previous timestep.

V. Improving Fast multirotor Descent and Landing by Learning the Ground Effect

To validate our methodology, we apply it to fast descent and landing of a multirotor[1]. As the vehicle approaches the landing plane, a ground effect from the interaction of the prop downwash and the landing surface becomes prominent. This effect induces added upward thrust on the drone, which can lead to poor tracking performance for control designs that rely on models which omit these fluid flow interactions.

A. Modeling and Problem Statement

To simplify the discussion, we consider a 1-dimensional *nominal* model of the multirotor's altitude dynamics, consisting of a point mass model having altitude and its derivative, $[p_z, \dot{p}_z]^T$, as states, mass m, and total thrust, T, as input:

$$\begin{bmatrix} \dot{p}_z \\ \ddot{p}_z \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_z \\ \dot{p}_z \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} T. \tag{16}$$

Using this model we design a nominal MPC as described in Section IV-B with the goal of reaching a fixed point of $0.05\,\text{m}$ above ground at zero velocity.

A nominal MPC stabilizes the drone to a fixed point, but uses more control effort and time to reach that point as a result of its simplified model. Importantly, the nominal dynamics model does not capture the ground effect. Our goal is to iteratively learn a better dynamics model (and associated MPC) that will improve speed and tracking performance in both the air and near-ground regimes.

B. Implementation and Experimental Details

Our experiments use the *Intel Aero RTF* Drone. Drone position is measured using an *OptiTrack* motion capture system and is fused with the drone's IMU (stock PX4 v1.8) to estimate the state. The diffeomorphism, h, is parameterized by a neural network and implemented with *PyTorch* [28], and the KEEDMD regression is implemented with elastic net regularization in *Scikit-learn* [29]. A dense form MPC-controller is implemented in Python using the QP solver *OSQP* [30], and commands are sent to the PX4 flight controller via *ROS*. All computation for learning and control is done on board the drone. Each neural network and MPC evaluation takes 5 ms, limiting us to 5 episodes as update rates below 60 hz lead to poor performance on our hardware.

[1]The code for learning and control is publicly available on `https://github.com/Cafolkes/keedmd`

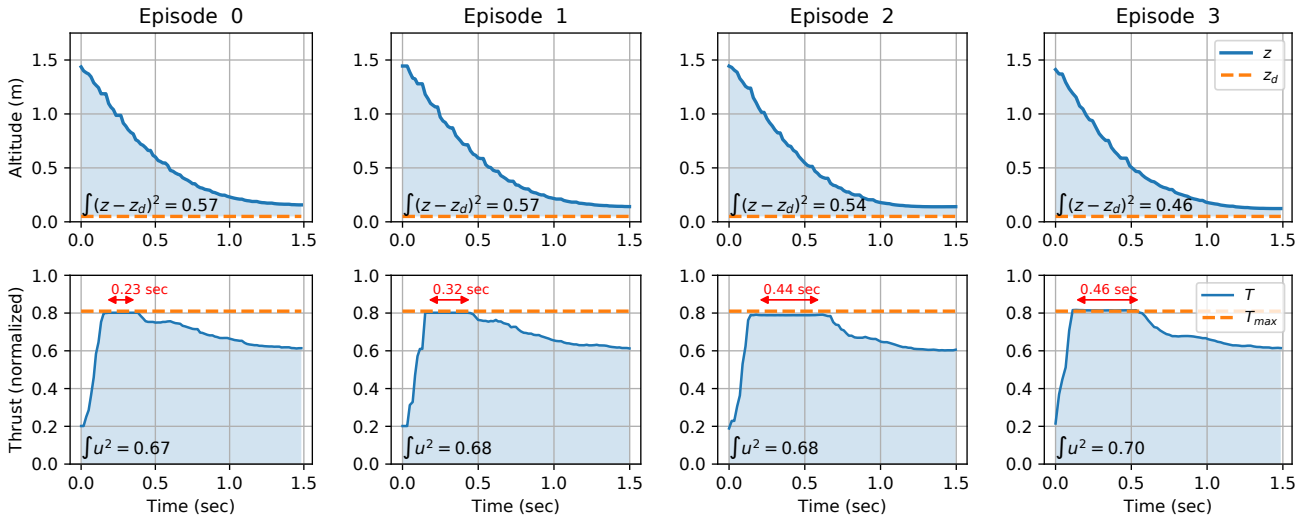

Fig. 3: Evolution of drone altitude p_z with accumulated error and control effort after each episode. Episode 0: baseline controller, Episode 1-3: performance after each episode of learning. Red arrows: duration the thrust constraint is active.

The experiment's key parameters are summarized in Table I.

We execute Algorithm 2 as discussed in Section IV on the drone for three episodes in each campaign. Each episode starts with 3 repetitions of the following: (1) the drone takes off and moves to an initial point under PX4 control; (2) the lifted controller takes over to stabilize the fixed point and hovers at that point for a second. After 3 repetitions, the drone lands under lifted control, fits the diffeomorphism and KEEDMD models, and repeats the episode. An additional landing sequence is executed to evaluate the performance of the current episode controller.

C. Results and Discussion

Figure 3 depicts the drone's trajectory and control effort under the nominal controller (Episode 0), and then final landing for three episodes of a single learning campaign. Episode 0 represents the nominal performance before learning, while episodes 1-3 show the learning effect. Tracking error is reduced by 19.3 percent by the end of the last episode while the total control effort increases 4.5 percent as a consequence of the chosen MPC penalty matrices. Importantly, the thrust constraint is rigorously satisfied, and this constraint is active for longer duration. As the system learns more accurate dynamic models, it relies more on the open-loop bang-bang characteristic, as would be expected from an optimal solution, and less from closed loop control. Less control effort is needed towards the end of the trajectory, indicating that our methodology captures the ground effect. The mean and standard deviation of five independent learning campaigns are reported in Fig. 4. The tracking performance improves in every episode. Furthermore, the methodology has low variance between campaigns.

TABLE I: Experiment Parameters

State error penalty, Q	$[10, 0.1]$	Min thrust, $u_{\min}$	0.3
Control penalty, R	1	Max thrust, $u_{\max}$	0.8
Min altitude, $x_{\min}$	0.05 m	Hover thrust, u_{hover}	0.66

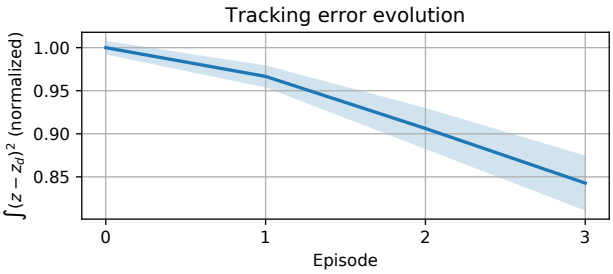

Fig. 4: Mean $\pm$ 1 standard deviation of tracking performance after each episode over 5 independent campaigns.

VI. Conclusions and Future Work

This paper presented a novel episodic method based on a Koopman eigenfunction framework to learn a robotic system's nonlinear dynamics, and learn a near optimal control strategy for given tasks. By using a Koopman approach, we are able to implement a real-time MPC framework for optimal system control during the learning process. The approach improves performance as it gathers more data, augmenting the controller to avoid constant actuation matrix limitations, while respecting state and control input bounds. A current limitation is the addition of a controller in each episode leading to prohibitive computational complexity as the number of episodes grows. Current work is addressing this by consolidating the controllers into a finite set, and by optimally choosing when to switch to the next episode, reducing the number of episodes needed.

Acknowledgement

This work has been supported in part by Raytheon Company and the DARPA Physics-infused AI program. The first author is grateful for the support of Aker Scholarship Foundation. The authors would like to thank Igor Mezic, Ryan Mohr, and Maria Fonoberova for helpful discussions.

REFERENCES

[1] G. M. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment," in *Proc. AAIA Guidance and Control Conf.*, 2007.

[2] M. Bangura and R. Mahoney, "Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor," in *Proc. Australasian Conf. on Robotics and Automation*, 2012.

[3] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian Process Dynamical Systems," in *Proc. Neural Information Processing Sytems*, 2006.

[4] Y. Duan, X. Chen, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," *arXiv*, 2016.

[5] J. García and F. Fernández, "A Comprehensive Survey on Safe Reinforcement Learning," *Journal of Machine Learning Research*, 2015.

[6] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks," in *Proc. AAAI*, 2019.

[7] Y. Lan and I. Mezić, "Linearization in the large of nonlinear systems and Koopman operator spectrum," *Physica D: Nonlinear Phenomena*, vol. 242, no. 1, pp. 42–53, 2013.

[8] A. Mauroy and J. Goncalves, "Linear identification of nonlinear systems: A lifting technique based on the Koopman operator," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6500–6505, IEEE, 2016.

[9] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos*, vol. 22, no. 4, 2012.

[10] K. Taira, S. L. Brunton, S. T. M. Dawson, C. W. Rowley, T. Colonius, B. J. Mckeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley, "Modal Analysis of Fluid Flows: An Overview," 2017.

[11] K. Taira, S. L. Brunton, S. T. M. Dawson, C. W. Rowley, T. Colonius, B. J. Mckeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley, "Modal Analysis of Fluid Flows: An Overview," 2017.

[12] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of Koopman eigenfunctions for control," no. April, 2017.

[13] J. L. Proctor, S. L. Brunton, and J. Nathan Kutz, "Generalizing Koopman Theory to Allow for Inputs and Control *," vol. 17, no. 1, pp. 909–930, 2018.

[14] R. Mohr and I. Mezić, "Construction of eigenfunctions for scalar-type operators via Laplace averages with connections to the Koopman operator," pp. 1–25, 2014.

[15] M. Korda and I. Mezić, "Learning Koopman eigenfunctions for prediction and control: the transient case," pp. 1–32, 2018.

[16] C. Folkestad, D. Pastor, I. Mezic, R. Mohr, M. Fonoberova, and J. Burdick, "Extended Dynamic Mode Decomposition with Learned Koopman Eigenfunctions for Prediction and Control," in *Proc. American Control Conf.*, Sep. (submitted) 2020.

[17] S. Ross, G. J. Gordon, J. A. Bagnell, and M. Learning, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," tech. rep.

[18] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[19] M. Nicole, "Real-time Model Predictive Control," 2011.

[20] C. Liu, H. Lu, and W.-H. Chen, "An explicit mpc for quadrotor trajectory tracking," in *2015 34th Chinese Control Conference (CCC)*, pp. 4055–4060, IEEE, 2015.

[21] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11773–11780, 2014.

[22] M. Abdolhosseini, Y. Zhang, and C. A. Rabbath, "An efficient model predictive control scheme for an unmanned quadrotor helicopter," *Journal of intelligent & robotic systems*, vol. 70, no. 1-4, pp. 27–38, 2013.

[23] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *2012 IEEE International Conference on Robotics and Automation*, pp. 279–284, IEEE, 2012.

[24] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural Lander: Stable Drone Landing Control using Learned Dynamics," *International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790, 2019.

[25] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2219, 2018.

[26] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018.

[27] B. J. Morris, M. J. Powell, and A. D. Ames, "Continuity and smoothness properties of nonlinear optimization-based feedback controllers," *Proceedings of the IEEE Conference on Decision and Control*, vol. 54rd IEEE, no. Cdc, pp. 151–158, 2015.

[28] A. Pazske, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, D. A. Lin, Zeming, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," tech. rep., 2017.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An Operator Splitting Solver for Quadratic Programs," *2018 UKACC 12th International Conference on Control, CONTROL 2018*, no. 1, p. 339, 2018.