Experimental Applications of the Koopman Operator in Active Learning for Control

Thomas A. Berrueta, Ian Abraham, and Todd Murphey

Abstract Experimentation as a setting for learning demands adaptability on the part of the decision-making system. It is typically infeasible for agents to have complete *a priori* knowledge of an environment, their own dynamics, or the behavior of other agents. In order to achieve autonomy in robotic applications, learning must occur incrementally, and ideally as a function of decision-making by exploiting the underlying control system. Most artificial intelligence techniques are ill-suited for experimental settings because they either lack the ability to learn incrementally or do not have information measures with which to guide their learning. This chapter examines the Koopman operator, its application in active learning, and its relationship to alternative learning techniques, such as Gaussian processes and kernel ridge regression. Additionally, examples are provided from a variety of experimental applications of the Koopman operator in active learning settings.

1 Introduction

Machine learning techniques describe complex patterns in data. Many learning methods, such as neural networks and convolutional networks, process large datasets in order to develop models that describe causal relationships between data. However, what happens when one must gather data as a part of a real-time process? How should one incorporate new data into models, and then leverage agency to further improve them? The answers to these questions form the basis of the robotic paradigm—*sense, plan, act*—and are at the heart of what defines autonomy.

In robotics, one is interested in endowing physical agents with the ability to reason about uncertainty and to *act* in order to achieve a goal. The experimental nature

Authors are with the Mechanical Engineering Dept. at Northwestern University, Evanston, IL

e-mail: tberrueta@u.northwestern.edu, i-abr@u.northwestern.edu, t-murphey@northwestern.edu

of this process necessitates learning frameworks capable of incremental adaptation, such that an agent is able to gather new information and use it to its advantage. We refer to the process in which an agent selectively seeks out new information relevant to better achieving a goal as *active learning*.

Not all machine learning techniques are well-suited for active learning. Batch learning processes requiring large amounts of training data are not useful in real-time settings where learning is occurring at all times. For active learning purposes, we need the ability to incrementally update models to avoid recomputing the learned model from scratch, which can be prohibitive. In order to develop principled approaches for incremental information acquisition, we need to quantify the model's information gain. We use the approximate Koopman operator as formulated in [38] and used in our earlier work [2] as a model of choice, and apply it in active learning settings.

While Koopman operators are the primary focus of this chapter, techniques such as generalized linear regression, kernel ridge regression, and Gaussian processes are often also applied as online learning tools in experiments. Specifying the theoretical and practical relationships between these methods is important to understanding the qualities of good active learning techniques. By developing a set of assumptions under which these methods are equivalent, one can transfer techniques developed for one method to another.

In Section 2, we describe the Koopman operator as a technique for active learning. In Section 3, we consider generalized linear regression, kernel ridge regression, and Gaussian processes as alternative techniques for experimental learning, and compare them to approximate Koopman operators, both in theoretical terms and computational implications. In Section 4, we illustrate the Koopman operator's use in experimental settings. In Section 5, we conclude with a discussion of what makes the Koopman operator a good system representation in active learning settings.

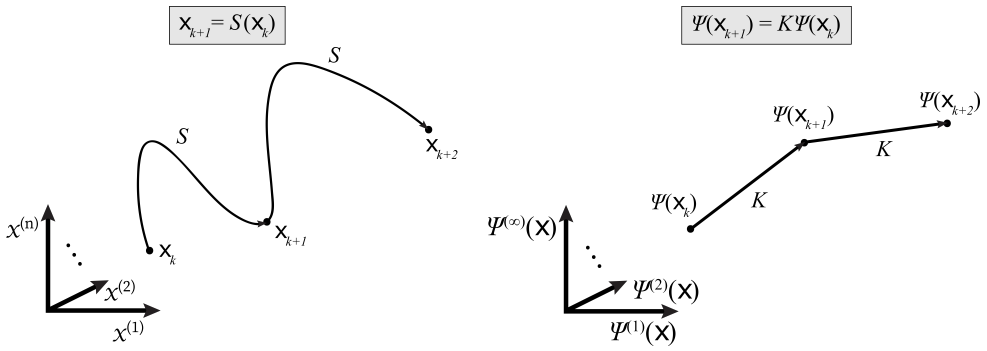
2 Koopman Operator

The Koopman operator is an infinite-dimensional linear operator capable of describing the time-evolution of any observable dynamical system [21]. We consider discrete-time dynamical systems described by

$$x_{k+1} = S(x_k) = x_k + \int_{t_k}^{t_k+\Delta t} F(x(\tau))d\tau, \tag{1}$$

where the dynamics evolve on a state-space manifold $\mathcal{M}$ such that $S : \mathcal{M} \to \mathcal{M}$. The right-hand side is a continuous-time system discretized with sampling interval Δt.

The Koopman operator is a linear operator that describes the time-evolution of basis functions instead of states. This shifts the system representation from state-space to an infinite-dimensional function space. Figure 1 shows the relationship



(a) Nonlinear state-space representation (b) Linear function-space representation

Fig. 1 Schematic of the relationship between a nonlinear dynamical system and its corresponding Koopman operator representation. Given a nonlinear system in n-dimensional state-space as shown in (a), the Koopman operator is a linear representation of the system in an infinite-dimensional function-space, shown in (b). The map Ψ lifts the original system onto an infinite-dimensional function-space. In this function-space, the Koopman operator is a linear transformation representing the system dynamics. Finite-dimensional approximations of the Koopman operator are denoted throughout this chapter as K. When referring to the infinite-dimensional operator, it is often denoted as U.

between a dynamical system $S(x)$ and its corresponding Koopman operator K. For a detailed analysis of the properties and theoretical applications of Koopman operators, we refer the reader to [7].

2.1 EDMD Approximation

The Koopman operator itself is not a learning tool; it is an alternative representation of dynamical systems. However, numerically synthesizing an approximation of this representation in finite dimensions from data is a learning problem. In order to implement Koopman operators in computational applications, one generates finite-dimensional approximations. While certain systems can be represented by an exact, closed-form, finite-dimensional Koopman operator, this is generally not the case [19]. Dynamic Mode Decomposition (DMD) [31] and Extended Dynamic Mode Decomposition (EDMD) [38] are methods developed for approximating Koopman operators in finite dimensions and are commonly applied in fluid dynamics as analytical tools [24].

EDMD synthesizes Koopman operator approximations by generating a mapping that spans a finite-dimensional subspace with a finite set of basis functions from data. For basis functions $\Psi(x) = [\psi_1(x), ..., \psi_N(x)]^T$, $s.t.\ \psi_i(x) \in \mathbb{R}$ and dataset $X = [x_1, ..., x_M]$, we can approximate a Koopman operator K to describe the evolution $\Psi(x_{k+1}) = K\Psi(x_k) + \varepsilon.\ s.t.\ \varepsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$ where $\mathcal{N}$ is a multivariate Gaussian distribution with mean and variance $\mathbf{0}, \sigma^2 \in \mathbb{R}^N$. The corresponding approximate

Koopman operator is given by the solution to the optimization

$$\min_{K} \frac{1}{2} \sum_{k=1}^{M-1} ||\Psi(x_{k+1}) - K\Psi(x_k)||^2 \tag{2}$$

which has closed form solution

$$K = AG^{\dagger}, \tag{3}$$

where † denotes the Moore-Penrose pseudoinverse and the individual matrix components are

$$\begin{aligned} G = G[M] &= \frac{1}{M} \sum_{k=1}^{M-1} \Psi(x_k)\Psi(x_k)^T \\ A = A[M] &= \frac{1}{M} \sum_{k=1}^{M-1} \Psi(x_{k+1})\Psi(x_k)^T. \end{aligned} \tag{4}$$

2.2 Incremental Updates

The closed-form solution in (4) is of particular interest because it can be modified for incremental updates [1]. We can define a set of difference equations describing an incremental update to the Koopman operator after a new observation

$$\begin{aligned} G[M+1] &= G[M] + \frac{1}{M+1}(\Psi(x_M)\Psi(x_M)^T - G[M]) \\ A[M+1] &= A[M] + \frac{1}{M+1}(\Psi(x_{M+1})\Psi(x_M)^T - A[M]), \end{aligned} \tag{5}$$

where we can calculate the updated Koopman operator with

$$K[M+1] = A[M+1](G[M+1])^{\dagger}. \tag{6}$$

This formulation does not scale in complexity with the number of measurements since the pseudoinverse computation complexity scales with respect to the number of basis functions considered. The difference equation (5) describes an implicit cumulative average of all measurements. However, it is possible to perform incremental updates to the Koopman operator using other difference equations, such as moving average filters, exponential moving average filters or Kalman filters [16, 14].

2.3 Formulation for Control

In order to apply the Koopman operator in experimental settings, we can formulate the operator for real-time control [6]. By considering control inputs as a part of the measurement vector, we can split the basis functions $\Psi(x,u)$ into $\Psi(x,u) = [\Psi_x(x)^T, \Psi_u(x,u)^T]^T$, where $\Psi_x(x) \in \mathbb{R}^{N_x}$ and $\Psi_u(x,u) \in \mathbb{R}^{N_u}$ such that $N = N_x + N_u$. As a result, the corresponding data-driven Koopman operator can be split into submatrices

$$K = \begin{bmatrix} K_x & K_u \\ \cdot & \cdot \end{bmatrix}. \tag{7}$$

Using these submatrices we can formulate the Koopman operator's time-evolution equation with control [2]

$$\Psi_x(x_{k+1}) = K_x \Psi_x(x_k) + K_u \Psi_u(x_k, u_k). \tag{8}$$

It is worth noting a couple special cases of (8). For the case in which $\Psi_u(x,u)$ is linear in u the equation becomes

$$\Psi_x(x_{k+1}) = K_x \Psi_x(x_k) + \hat{K}_u u_k, \tag{9}$$

where $\hat{K}_u = \frac{\partial \Psi_u(x_k,u_k)}{\partial u} K_u$. Then for the case in which $\Psi_u(x,u)$ is linear in u and does not depend on the state the equation is

$$\Psi_x(x_{k+1}) = K_x \Psi_x(x_k) + K_u u_k. \tag{10}$$

This formulation in (10) of the Koopman operator for control systems enables us to easily incorporate it into classical control schemes such as optimal control, or other model-predictive frameworks [22]. Alternative data-driven formulations of the Koopman operator for control have been described in the work of [19].

In optimal control, we seek to specify control laws that best drive systems towards a desired goal state with respect to an objective. For a discrete-time linear quadratic (LQ) control problem with goal state x_d, we can define an objective function of the following form over the iterate k

$$J = \sum_{k=0}^{\infty} (\Psi_x(x_k) - \Psi_x(x_d))^T Q (\Psi_x(x_k) - \Psi_x(x_d)) + u_k^T R u_k, \tag{11}$$

where Q and R are positive semi-definite matrices that specify weights on system states and control effort, respectively. Deriving an optimal solution to this LQ problem leads to a feedback law

$$u_{LQ_K} = -F_{LQ_K}(\Psi_x(x) - \Psi_x(x_d)) \tag{12}$$

such that the optimal feedback gain F_{LQ_K} is

$$F_{LQ_K} = (R + K_u^T P K_u)^{-1} K_u^T P K_x, \tag{13}$$

where P is the solution to the discrete-time algebraic Ricatti equation. Through this process, we can solve optimal control problems with data-driven Koopman models [6, 19].

Additionally, given Koopman operator dynamics of the same form as (10), we note that system derivatives with respect to basis functions of state and control are linear:

$$\frac{d}{d\Psi_x}\big(\Psi_x(x_{k+1})\big) = K_x, \quad \frac{d}{du}\big(\Psi_x(x_{k+1})\big) = K_u. \tag{14}$$

The quality of a data-driven model is dependent on the amount of information captured about the underlying system. In order to determine how much information is captured by system models, we must develop a means of quantifying information with respect to the model.

2.4 Information Measures

Active learning is a process by which an agent can acquire more information about a task and improve its performance. Through experimental learning, we are interested in improving our system models such that we maximize task performance. In order to actively acquire information, we must develop information measures with respect to our system model. Fisher information provides a means of measuring the amount of information that a random variable carries about parameters [10]. Given a Koopman operator K_x, we model each of its elements as a normally distributed parameter $K_x^{(i,j)} \sim \mathcal{N}(\mu_{ij}, \Sigma_{ij}^2)$, and compute the Fisher information

$$I(\Psi_x(x)) = \frac{\partial \Gamma^T}{\partial \bar{k}} \Sigma^{-1} \frac{\partial \Gamma}{\partial \bar{k}}, \tag{15}$$

where $\bar{k} = \{K_x^{(i,j)} | i, j \in \{1,...,N_x\} \times \{1,...,N_x\}\}$, $\Gamma \in \mathbb{R}^{N_x}$ is the measurement model defined by $\Gamma = K_x \Psi_x(x)$, and $\Sigma \in \mathbb{R}^{N_x \times N_x}$ is assumed to be constant. The matrix Σ is the measurement covariance that can be empirically estimated from sample data *a priori*. The matrix $\frac{\partial \Gamma}{\partial \bar{k}} \in \mathbb{R}^{N_x \times (N_x \cdot N_x)}$ encodes measurement sensitivities to all elements of the Koopman operator. The sensitivity matrix takes the form

$$\frac{\partial \Gamma}{\partial \bar{k}} = \begin{bmatrix} \Psi_x(x)^T & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \Psi_x(x)^T & ... & \mathbf{0} \\ \vdots & & \ddots & \\ \mathbf{0} & ... & & \Psi_x(x)^T \end{bmatrix}, \tag{16}$$

where $\mathbf{0} \in \mathbb{R}^{1 \times N_x}$. We now optimize the trace of the Fisher information matrix, also known as T-optimality [34], to approximate information gain

$$\text{tr}(I(\Psi_x(x))) = \sum_{i=1}^{N_x} ||\Psi_x(x)||^2_{\Sigma_{ii}^{-1}} \ . \tag{17}$$

We introduce (17) into a control objective by assigning a cost q to the inverse of the Fisher information

$$J = \sum_{k=1}^{\infty} q \ \text{tr}(I(\Psi_x(x_k)))^{-1} + ||\Psi_x(x_k) - \Psi_x(x_d)||^2_Q + ||u_k||^2_R \ . \tag{18}$$

Using the cost functional in (18) we synthesize information maximizing trajectories and continuously improve our Koopman operator representation of the dynamics, as shown in [1]. The simplicity of the Koopman operator's measurement model Γ makes it easy to develop information measures with respect to the generated model.

2.5 Formulation for Hybrid Systems

Robotic systems often have discontinuous dynamics because of impacts and intermittent contact, leading to hybrid dynamics. Hybrid system dynamics are defined piecewise, where an indicator function determines which hybrid mode will evolve the current state. These dynamics are formulated as

$$S(x) = \begin{cases} S_1(x), & \text{if } \Phi(x) = 1 \\ \vdots & \vdots \\ S_i(x), & \text{if } \Phi(x) = i \\ \vdots & \vdots \\ S_B(x), & \text{otherwise} \end{cases} \tag{19}$$

where each $S_i(x)$ represents the dynamics of states $x \in \mathbb{R}^n$ at mode $\Phi(x) = i$. Additionally, at each hybrid mode boundary there is a discontinuous map between modes. Assuming that $\Phi(x)$ is known *a priori*, it is possible to compute each Koopman operator K_i to represent each hybrid mode. We can specify a hybrid Koopman operator using

$$\overline{K}(x), \overline{\Psi}(x) = \begin{cases} K_1(x), \Psi_1, & \text{if } \Phi(x) = 1 \\ \vdots & \vdots \\ K_i(x), \Psi_i, & \text{if } \Phi(x) = i \\ \vdots & \vdots \\ K_B(x), \Psi_B, & \text{otherwise} \end{cases} \tag{20}$$

where we characterize the discontinuous jumps between hybrid modes by collecting data at the mode boundaries and using $\Psi_i^*(x_k^+) = \Psi_i^*(x_k^-)K_i^*$. Given that these jumps in state often take place over very small time scales, it may be difficult to

gather enough data to provide a robust Koopman operator for that mapping. Active excitation methods might be necessary in order to properly characterize transitions. Alternative treatments of the Koopman operator have looked into modeling and analyzing switched [29], hybrid [15], and distributed systems [17].

The Koopman operator's linearity, ability to update incrementally, and ease of generating information measures make it a good choice of data-driven model representation for experimental settings. However, the operator's performance is dependent on the suitability of the chosen basis functions, which may not be known ahead of time.

Though the model representation is different, data-driven Koopman operator synthesis, as formulated in EDMD, solves a similar least-squares optimization as alternative regression models, such as generalized linear regression, kernel ridge regression, and Gaussian processes. Characterizing the conditions under which these models are equivalent to one another can help expand understanding of both Koopman operators synthesis and alternative learning methods. Under certain sets of assumptions, methods developed for Koopman operators synthesized via EDMD apply to the alternatives, and *vice versa*.

3 Alternative Methods

While the Koopman operator is neither a function approximation technique nor a regression tool, EDMD, as formulated in Section 2.1, is both [38]. In Section 2 we proposed that the EDMD approximation of the Koopman operator is well-suited for active learning. In principle, other function approximation techniques can be applied to generate finite Koopman operators. However, EDMD is the most common Koopman operator synthesis technique, and as such it is of interest to explicitly relate it to alternative machine learning techniques.

We show that under certain assumptions generalized linear regression, kernel ridge regression, and Gaussian processes solve the same underlying optimization problem as EDMD and can be formulated similarly. By understanding these relationships, we can apply techniques formally developed for one method to others, and *vice versa*. Through these equivalences we can extend the active learning techniques developed for the Koopman operator in Section 2 to other methods. We first provide a review of generalized linear regression, kernel ridge regression, and Gaussian processes, then establish the relationships between them.

3.1 Generalized Linear Regression

Linear regression and online linear regression have long been applied for statistical analysis of sequential models [32]. Linear regression models are typically limited to describing a function $y = f(x),\ y,x \in \mathbb{R}^S$ as linear combinations of in-

puts $y = W^T x$, where we find $W \in \mathbb{R}^{S \times S}$ to minimize an error metric. Generalized linear regression extends this approach to consider nonlinear functions of state, such that we are performing a linear regression in a given feature space defined by the basis functions, where the features are the basis functions. Given a dataset with inputs $X = [x_1, ..., x_M]$, outputs $Y = [y_1, ..., y_M]$, and a set of basis functions $\Psi(x) = [\psi_1(x), ..., \psi_N(x)]^T \in \mathbb{R}^N$, we consider the set of candidate functions defined by linear combinations of the basis functions subject to zero-mean Gaussian noise ε with variance σ^2

$$y = W^T \Psi(x) + \varepsilon. \tag{21}$$

We can minimize over the least-squares error functional

$$\min_W \frac{1}{2} \sum_{k=1}^{M-1} ||y_k - W^T \Psi(x_k)||^2 \tag{22}$$

to obtain a closed-form solution using maximum likelihood estimation [5]. We define the design matrix as

$$\Phi = [\Psi(x_1), ..., \Psi(x_M)]^T \in \mathbb{R}^{M \times N} \tag{23}$$

which allows us to express the closed-form solution of (22) as

$$W = (\Phi^T \Phi)^{-1} \Phi^T Y, \tag{24}$$

where $Y = [y_1, ..., y_M]^T$. We note that (24) is equivalent to

$$W = \sum_{k=1}^{M-1} (\Psi(x_k) \Psi(x_k)^T)^{-1} (\Psi(x_k)^T y_k). \tag{25}$$

We also note that for autoregressive models where $y_k = \Psi(x_{k+1})$

$$\Psi(x_{k+1}) = W^T \Psi(x_k). \tag{26}$$

3.2 Kernel Ridge Regression

Kernel ridge regression reframes the generalized linear regression problem by considering infinite-dimensional feature spaces specified by kernels. Features are observed properties or characteristics of a phenomenon [5]. A feature space is then the space formed by all features considered. A kernel is a generalized inner product between vectors in a feature space. Finite-dimensional kernels can be specified directly in terms of a choice of basis functions

$$k(x, x') = \langle \Psi(x), \Psi(x') \rangle, \tag{27}$$

where the basis functions are a nonlinear transformation into a feature space. Kernels can implicitly define inner products in infinite-dimensional spaces without having to directly project coordinates into the feature space. The most common infinite-dimensional kernel of choice is the radial basis function kernel, also known as the Gaussian kernel [5]

$$k(x,x') = e^{-\frac{1}{2\sigma^2}||x-x'||^2}. \tag{28}$$

The sense in which the Gaussian kernel represents an inner product in an infinite-dimensional space can be understood by examining its Taylor series expansion. The Gaussian kernel is shown below as an infinite-dimensional polynomial kernel, with $\sigma^2 = \frac{1}{2}$ for simplicity:

$$\begin{aligned} k(x,x') &= e^{-||x-x'||^2} \\ &= e^{-(x^2+2(x^Tx')+(x')^2)} \\ &= (e^{-x^2})(e^{-(x')^2}) \sum_{n=0}^{\infty} \frac{(2x^Tx')^n}{n!} \\ &= C(x,x') \sum_{n=0}^{\infty} \frac{2^n}{n!} \langle x,x' \rangle^n, \end{aligned} \tag{29}$$

where $C(x,x')$ is some constant value dependent on x and x' that does not affect the Taylor expansion. Equation (29) shows that the Gaussian kernel is an inner product over the infinite-dimensional space of polynomial functions, where polynomial functions of the input vector are considered as features.

Ridge regression, also known as L_2-norm regularized generalized linear regression, extends the least-squares residual minimization problem described in (22) by including a regularization term over the model weights. Solutions with low weight magnitudes are given preference to prevent overfitting of the model. For a dataset with inputs $X = [x_1, ...x_M]$ and outputs $Y = [y_1, ...y_M]$, the ridge regression optimization is given by

$$\min_W \frac{1}{2} \sum_{k=1}^{M-1} ||y_k - W^T \Psi(x_k)||^2 + \frac{\lambda}{2} ||W||_F^2, \tag{30}$$

where $\lambda \geqslant 0$ is a regularization parameter specifying a cost on the magnitude of the model weights, and $||\cdot||_F$ represents the Frobenius matrix norm.

Using the design matrix Φ in (23), we can write down a closed-form solution to the minimization in (30)

$$W = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T Y. \tag{31}$$

We refer to this solution as the primal solution to the ridge regression estimation problem.

Kernel ridge regression is based on an alternative representation of the solution to the ridge regression problem that enables application with infinite-dimensional kernels. The primal solution to the ridge regression problem scales in complexity

with the number of basis functions used, which makes it unable to consider high-dimensional feature spaces. However, we can manipulate (31) to get an equivalent representation, which we will refer to as the dual solution

$$W = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}Y. \tag{32}$$

We define $B = (G_r + \lambda I)^{-1}Y$ where G_r is the Gram matrix, such that $W = \Phi^T B$. The Gram matrix is a matrix containing all inner products between points in a dataset as specified by a given kernel $k(x,x')$, such that $G_r^{(i,j)} = k(x_i,x_j),\ \forall i,j \in \{1,...,M\}$. In this case, $G_r = \Phi\Phi^T$ due to the choice of kernel in (27). If we consider W as the primal weight matrix, we can refer to B as the dual weight matrix and use it to formulate our estimate as a function of the kernel choice [25]

$$\begin{aligned}\hat{y} &= \Psi(x')^T W &(33)\\ &= \Psi(x')^T\Phi^T B\\ &= \mathbf{k}(x')^T B\\ &= \mathbf{k}(x')^T(G_r + \lambda I)^{-1}Y, &(34)\end{aligned}$$

where $\mathbf{k}(x') = \Phi\Psi(x') = [k(x_1,x'),...,k(x_M,x')]^T$. The kernel ridge regression function prediction scales in complexity with the number of data points, and as such is not suitable for incremental learning. Equation (34) enables us to analyze the models in infinite-dimensional feature spaces using kernels, which is beneficial in different settings. This kernel-based representation is how kernel ridge regression is most often applied. However, kernel ridge regression can also be formulated recursively and applied in online learning [9, 12].

3.3 Gaussian Process Regression

Until this point, we have only considered non-probabilistic models for regression in active learning. In some settings, randomness and uncertainty are inherently present in state measurements and as such require models that incorporate uncertainty into their predictions. Stochastic processes, such as Gaussian processes, specialize in describing systems that probabilistically evolve over time [14].

A Gaussian process is a stochastic process where any collection of random variables drawn are jointly Gaussian. This is to say that any finite set of random variable observations will be distributed according to $f(x_1,...,x_M) \sim \mathcal{N}(\mu,\Sigma),\ s.t\ \mu \in \mathbb{R}^M,\ \Sigma \in \mathbb{R}^{M\times M}$. We specify a Gaussian process by defining its mean and covariance functions, $m(x)$ and $k(x,x')$. Covariance functions describe a similarity measure between features in an infinite-dimensional function space, and are equivalent to positive-definite kernel functions [30]. Given a Gaussian process $f(x)$

$$
\begin{aligned}
f(x) &\sim \mathcal{GP}(m(x), k(x,x')) \\
m(x) &= \mathbb{E}[f(x)] \\
k(x,x') &= \mathbb{E}[(f(x)-m(x))^T (f(x')-m(x'))].
\end{aligned} \tag{35}
$$

Gaussian process regression specifies an approximated function by expressing a predictive distribution over a function space based on observations of realized random variables. This predictive distribution $p(f(x')|X,Y)$ is also known as a posterior, and can be computed given inputs $X = [x_1, ..., x_M]$ and outputs $Y = [y_1, ..., y_M]$ through a Bayesian inference framework

$$
\begin{aligned}
\text{posterior} &= \frac{\text{prior} \times \text{likelihood}}{\text{marginal likelihood}} \\
p(f(x)|X,Y) &= \frac{p(f(x))p(Y|X,f(x))}{\int p(Y|f(x),X)p(f(x)|X)df(x)}.
\end{aligned} \tag{36}
$$

By exploiting properties of the Gaussian distribution one can avoid explicitly computing the posterior in (36). We use the fact that conditioning and marginalizations of Gaussian distributions are other Gaussians with closed-form solutions [30]. Without imposing any prior belief on the function space this problem is ill-posed. We will consider processes of the form $y_k = f(x_k) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. As a prior, we restrict the set of functions considered to $y = W^T \Psi(x) + \varepsilon$ such that $W \sim \mathcal{N}(0, \Sigma_W)$ with zero mean for notational simplicity. We can see that the mean and covariance functions are then

$$
\begin{aligned}
m(x) &= \mathbb{E}[f(x)] = \mathbb{E}[W^T]\Psi(x) + \mathbb{E}[\varepsilon] = 0 \\
k(x,x') &= \mathbb{E}[f(x)^T f(x')] = \Psi(x)^T \mathbb{E}[WW^T]\Psi(x') + \mathbb{E}[\varepsilon^T \varepsilon] = \Psi(x)^T \Sigma_W \Psi(x') + \sigma^2.
\end{aligned} \tag{37}
$$

Given a design matrix as defined in (23), one can use the mean and covariance functions to define a Gaussian processes $f(x) \sim \mathcal{GP}(m(x), k(x,x'))$. To generate a predictive distribution, we then define a joint Gaussian distribution between the input dataset's design matrix Φ and a new observation $\Psi(x')$

$$
\begin{bmatrix} f(X) \\ f(x') \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{k}(\Phi,\Phi) & \mathbf{k}(\Phi,\Psi(x')) \\ \mathbf{k}(\Phi,\Psi(x'))^T & k(\Psi(x'),\Psi(x')) \end{bmatrix}\right), \tag{38}
$$

where the design matrix autocovariance is $\mathbf{k}(\Phi,\Phi) \in \mathbb{R}^{M \times M}$, and the covariance between the design matrix and the new observation $\mathbf{k}(\Phi,\Psi(x')) \in \mathbb{R}^M$. We then express these covariances in closed form

$$
\begin{bmatrix} f(X) \\ f(x') \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \Phi\Sigma_W\Phi^T + \sigma^2 I & \Phi\Sigma_W\Psi(x') + \sigma^2 \mathbf{1} \\ (\Phi\Sigma_W\Psi(x'))^T + \sigma^2 \mathbf{1}^T & \Psi(x')^T\Sigma_W\Psi(x') + \sigma^2 \end{bmatrix}\right), \tag{39}
$$

where $\mathbf{1} \in \mathbb{R}^M$. Equation (39) allows us to calculate a predictive posterior distribution for new observations of $f(\cdot)$ by marginalizing over the design matrix Φ and

outputs Y [36]. The predictive posterior distribution is then of the form

$$\begin{aligned} f(x')|\Phi,Y &\sim \mathcal{N}(\mathbb{E}[f(x')],\mathbb{V}[f(x')]) \\ \mathbb{E}[f(x')] &= \Psi(x')^T\Sigma_W\Phi^T(\Phi\Sigma_W\Phi^T+\sigma^2 I)^{-1}Y \\ \mathbb{V}[f(x')] &= \Psi(x')^T\Sigma_W\Psi(x') - \Psi(x')^T\Sigma_W\Phi^T(\Phi\Sigma_W\Phi^T+\sigma^2 I)^{-1}\Phi\Sigma_W\Psi(x'). \end{aligned} \tag{40}$$

This posterior distribution enables us to generate predictions for new observations, as well as specify a covariance for each point [30].

Although we have constructed our derivation using a finite-dimensional basis representation, Gaussian processes are typically implemented using infinite-dimensional kernels. By following the same procedure shown in Section 3.2, kernelization is trivial, and results in the following predictive distribution in terms of a specified kernel:

$$\begin{aligned} f(x')|\Phi,Y &\sim \mathcal{N}(\mathbb{E}[f(x')],\mathbb{V}[f(x')]) \\ \mathbb{E}[f(x')] &= m(x') + \mathbf{k}(x')^T(G_r+\sigma^2 I)^{-1}(Y-m(X)) \\ \mathbb{V}[f(x')] &= k(x',x') - \mathbf{k}(x')^T(G_r+\sigma^2 I)^{-1}\mathbf{k}(x'), \end{aligned} \tag{41}$$

where we have added the mean function $m(x)$ for generality. The case in which $m(x)=0$ is referred to as a *centered* Gaussian process. This kernelized representation of the Gaussian process is not well-suited for active learning settings. In order to overcome this limitation, there has been work in deriving local models for real-time applications [27].

3.4 Relationships Between Learning Techniques

Now that we have reviewed the foundations of generalized linear regression, kernel ridge regression, and Gaussian processes, we can examine their relationships to EDMD approximations of Koopman operators, as well as to one another. We develop assumptions for which these techniques are equivalent to one another.

3.4.1 Generalized Linear Regression

For finite-dimensional bases, generalized linear regression is most closely related to EDMD. By limiting the functions we are willing to model to autoregressive functions, we can make the substitution $y_k = \Psi(x_{k+1})$ in (25). Then, if we note that $A^\dagger = A^{-1}$ when A is square, we can restate (25) in relation to the Koopman operator matrix components in (4)

$$
\begin{aligned}
W &= \sum_{k=1}^{M-1} (\Psi(x_k)\Psi(x_k)^T)^\dagger (\Psi(x_k)^T \Psi(x_{k+1})) \\
&= ((G[M])^\dagger)^T (A[M])^T = (A[M](G[M])^\dagger)^T \\
&= K^T. \qquad (42)
\end{aligned}
$$

Thus, generalized linear regression models are equivalent to finite-dimensional Koopman operators synthesized through EDMD for the same set of basis functions. This should not be surprising since Koopman operator synthesis via DMD [31, 35] and EDMD [38] is formulated as a regression problem.

The correspondence between these techniques indicates that all methods developed in Section 2 apply to generalized linear regression, and *vice versa*. This means that generalized linear regression models can be applied in control, represent hybrid systems, update incrementally, and have information measures easily generated, which makes them suitable for experimental settings.

3.4.2 Kernel Ridge Regression

Kernel ridge regression models are closely related to generalized linear regression models, and consequently Koopman operators. Ridge regression solves a slightly different optimization problem than EDMD, as shown in (30). However, for the case of $\lambda = 0$ the ridge regression optimization is equivalent to that of generalized linear regression and EDMD, as shown in (22).

While the kernel ridge regression model estimate for infinite-dimensional kernels in (34) cannot be directly related to Koopman models, we have shown that the solution for finite-dimensional kernels $k(x,x') = \langle \Psi(x), \Psi(x') \rangle,\ s.t.\ \Psi(x) \in \mathbb{R}^n$ in (32) is derived from (31) in Section 3.2. For $\lambda = 0$, (31) is equivalent to the generalized linear regression solution, which we have shown in the previous section to be the same as that of EDMD. This is to say that the kernel ridge regression estimate can be equivalent to the Koopman operator's for a regularization coefficient of zero, and for finite-dimensional kernels. In this limited representation, kernel ridge regression models are equivalent to approximate Koopman operators and can inherit methods developed for the latter.

It is important to note that kernel ridge regression without infinite-dimensional kernels or regularization is not how the technique is typically implemented, and does not take advantage of the compactness of infinite-dimensional kernel representations. Since typical applications of kernel ridge regression scale in complexity with the number of data observed, it is ill-suited for experimental learning. However, the regularization coefficient does not prevent online computation, which means that ridge regression without infinite-dimensional kernels can be applied as a real-time learning tool for Koopman operators in settings where model sparsity is desirable.

3.4.3 Gaussian Processes

Despite Gaussian processes being stochastic, they are closely related to kernel ridge regression. By examining the Gaussian processes' mean function in (39) we see that it is equivalent to the kernel ridge regression prediction in (33) given that we set $\sigma^2 = \lambda$, and $\Sigma_W = I$, with W as defined in (32). In order for the equivalence to hold between typical kernel ridge regression models and Gaussian processes, we ignore the process covariance estimate and sequential Bayesian model updates. However, recent work in Bayesian kernel ridge regression models has shown that they are equivalent to Gaussian processes when the Gaussian process covariance estimate is fixed [37].

The correspondence between Gaussian process regression models and Koopman operator synthesis follows from EDMD's relationship to kernel ridge regression as described in the previous section. By setting $\sigma^2 = 0$ and $\Sigma_W = I$ the mean estimate from the Gaussian process regression matches the EDMD solution. However, by doing this we remove all non-determinism from the function approximation which makes it fundamentally *not* a Gaussian process. Typical implementations of Gaussian processes scale in complexity with the number of data observed, and as such are not suitable for settings that demand incremental learning. Nonetheless, one can generate information measures for Gaussian processes with stationary mean and covariance functions [11].

Alternatively, both generalized linear regression and kernel ridge regression can be derived through a Bayesian inference framework [5], which implies that EDMD can itself be derived in non-deterministic settings and be subject to a Bayesian treatment. It would be of interest to show a direct correspondence between Bayesian representations of the Koopman operator and Gaussian processes because of their relationship to neural networks. Given a Gaussian prior over model weights, [26] showed that a single-layer neural network with infinite weights approximates a Gaussian process as a result of the central limit theorem. Recently, this work was extended to describe the relationship between Gaussian processes and deep neural networks in [23].

3.4.4 Summary

The assumptions under which generalized linear regression, kernel ridge regression, and Gaussian processes produce the same solutions to an estimation problem as EDMD are summarized in Table 1. Approximate Koopman operators as formulated in Section 2.1 are equivalent to generalized linear regression models. Kernel ridge regression models can be made equivalent to such a Koopman operator model under a set of simplifying assumptions for finite kernel representations. Finally, the relationship between Gaussian processes and EDMD approximations of Koopman operators follows from its correspondence with kernel ridge regression in deterministic settings.

	Generalized Linear Regression	Kernel Ridge Regression	Gaussian Processes
Koopman Operators (EDMD)	$K = W^T$	$\lambda = 0$ $k(x,x') = \langle \Psi(x), \Psi(x') \rangle$	$\sigma^2 = 0,\ \Sigma_W = I$ $k(x,x') = \langle \Psi(x), \Psi(x') \rangle$ Ignore covariance estimate

Table 1 Assumptions under which each alternative method presented in Section 3 is equivalent to the EDMD estimate of a finite Koopman operator. As formulated, generalized linear regression models are equivalent to the synthesized Koopman operator matrix. The kernel ridge regression prediction is equivalent to that of EDMD when the regression regularization coefficient is 0, and only for finite-dimensional kernels. The mean function estimate of a Gaussian process is equivalent to an EDMD Koopman operator prediction when $\sigma^2 = 0$, $\Sigma_W = I$, for finite-dimensional kernels.

Although Gaussian processes and kernel ridge regression are often applied in real-time settings, in their typical formulations they are unable to be updated incrementally and are ill-suited for active learning problems. However, generalized linear regression models are compatible with experimental settings because of their direct correspondence with EDMD. The correspondences between methods shown throughout this section imply that there should exist settings in which learning techniques such as Gaussian processes and deep neural networks can be made to learn incrementally and with respect to an information measure.

3.5 Example

Though the methods presented in this section are formulated through alternative means, they all solve similar optimization problems. The primary difference between these methods is representation, which can lead to *the formal optimization being the same, but the implementation being different*. Under a set of assumptions we are able to relate the model representations of kernel ridge regression, Gaussian processes, and EDMD. Although this is technically the case, these assumptions do not represent typical use cases for these methods.

As a comparison between standard implementations of these methods, we use each one to synthesize models of a double pendulum system's dynamics from data. We do not implement generalized linear regression because we showed it to produce an equivalent model representation to the EDMD solution in Section 3.4.1. We collected a training dataset consisting of a single $4s$ trajectory taken from the double pendulum free dynamics sampled at $100Hz$ with an initial condition of $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0.8, 0, 0, 0)$. We use the training dataset to instantiate a Koopman operator, kernel ridge regression model, and a Gaussian process model. The Koopman operator model is calculated using a second-order polynomial basis of the double pendulum states, while both the kernel ridge regression model and the Gaussian process use the Gaussian kernel shown in (28). The regularization and variance parameters were selected by the Scikit-Learn software package [28] via Bayesian

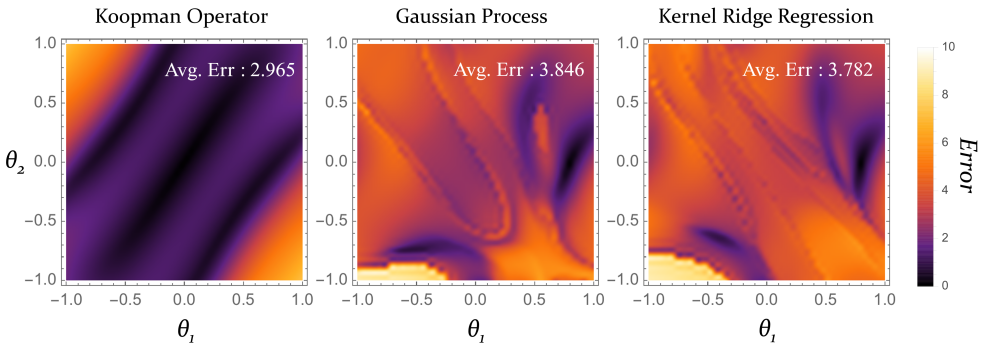


Fig. 2 Performance comparison between Koopman operators synthesized with EDMD, Gaussian process models, and kernel ridge regression models. Each model was trained on the same dataset collected from a $4s$ trajectory of the free dynamics of a double pendulum system with initial condition $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0.8, 0, 0, 0)$ sampled at $100Hz$. Then, we calculated the integrated mean squared error over a $3s$ prediction from each model of the double pendulum dynamics at each (θ_1, θ_2) location over the $\{(\theta_1, \theta_2) : [-1, 1] \times [-1, 1]\}$ domain. The Gaussian process and kernel ridge regression models were implemented using the Gaussian kernel in (28). The Koopman operator was synthesized using a second order polynomial basis. We note that the operator produces an estimate of the dynamics with lower error over the domain. Standard implementations of each model were used so as to compare performance under typical usage.

hyperparameter optimization [33]. Each model is then used to predict double pendulum trajectories for a horizon of $3s$ from each (θ_1, θ_2) initial condition over the $\{(\theta_1, \theta_2) : [-1, 1] \times [-1, 1]\}$ domain, with zero initial velocity. Then, we calculate the integrated mean square error between each model's prediction and the actual dynamics over the entire horizon for each initial condition.

Figure 2 depicts the results of the comparison. The Koopman operator prediction has the lowest average error of the three models. We observe that both the Gaussian process and kernel ridge regression prediction errors are lowest at the initial conditions of the training trajectory $(\theta_1, \theta_2) = (0.8, 0)$, while the Koopman operator model generalizes more easily over the domain. The purpose of this comparison is to show that while these methods can be shown to be closely related, standard implementations will produce very different results.

4 Experimental Applications

In this section, we demonstrate experimental applications of the Koopman operator in a variety of examples. We highlight applications in which active learning enables rapid system identification through information-driven incremental model updates. Additionally, we show how the Koopman operator's ability to represent hybrid systems enables the data-driven modeling of finite automata in experimental settings.

Fig. 3 Sphero SPRK robot used for active learning of system dynamics on sand.

4.1 Learning Sphero SPRK Dynamics in Sand

We are interested in demonstrating the use of a Koopman operator applied to learning complex nonlinear dynamics in a real-time experimental setting. We use the Sphero SPRK shown in Fig. 3, which is a programmable robotic toy shaped like a small ball that can be teleoperated using a controller or app. It is actuated through an internal mechanism designed for rolling on flat ground. If we change the system dynamics by switching the rolling surface to sand the SPRK is incapable of rolling with its default controller. To enable the SPRK to roll on sand, we use the Koopman operator for active identification of the robot's dynamics on sand via real-time information maximization as described in Section 2.4, and shown in [1].

We use a nonlinear model-predictive controller known as Sequential Action Control (SAC) [3] to generate information maximizing trajectories with respect to the information objective (18), which we use to update our Koopman operator model in real time using (6). Over a $20s$ period, we synthesize information maximizing trajectories in order to characterize system dynamics on sand. After this period, we switch objectives to a tracking objective such as (11) where we attempt to trace a figure 8 trajectory in a sandbox while continuing to update the Koopman operator. We compare the actively learned Koopman operator performance to a passive Koopman operator. The passive Koopman operator was computed from $20s$ of data from tracing the figure 8 trajectory with open-loop control.

The experiment consists of a rectangular sandbox, with an Xbox Kinect mounted overhead equipped with OpenCV providing odometry information. The system states collected are $\mathbf{x} = [x, y, \dot{x}, \dot{y}]^T$ where (x, y) are coordinates in the Kinect field of vision, and we define our control signals within this representation as $\mathbf{u} = [u_x, u_y]^T$. The set of basis functions used to model the SPRK robot were

$$\begin{aligned} \Psi_x(\mathbf{x}) &= [x, y, \dot{x}, \dot{y}, 1, \dot{x}^2, \dot{y}^2, \dot{x}^2\dot{y}, ..., \dot{x}^3\dot{y}^3]^T \in \mathbb{R}^{18} \\ \Psi_u(\mathbf{u}) &= [u_x, u_y]^T \in \mathbb{R}^2, \end{aligned} \tag{43}$$

which are linear combinations of third order polynomial basis functions of the velocity states. The basis functions of control were chosen to be linear in $\mathbf{u}$ and not dependent on state.

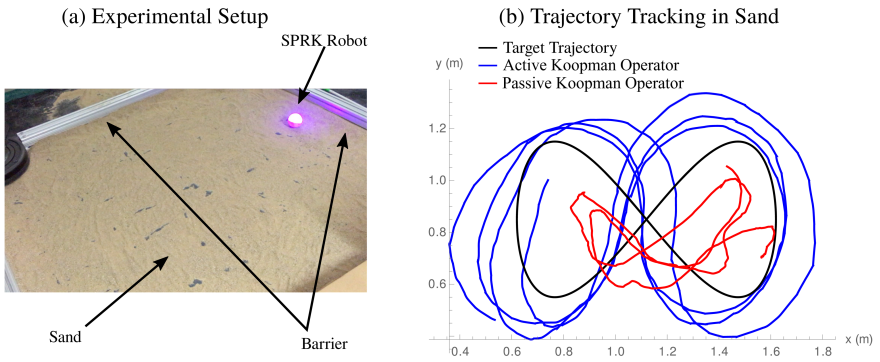


Fig. 4 Experimental application of active learning with Koopman operators for online learning of the Sphero SPRK dynamics on sand. (a) Experimental setup consisting of a boxed area covered in sand, with an overhead Xbox Kinect using OpenCV reporting position information. (b) Results from trajectory tracking experiments in sand. The Sphero SPRK was used to track a figure 8 trajectory (black), using both an active learning approach with a continuously updating Koopman operator (blue), and a pre-computed passive Koopman operator (red). While the fixed model establishes a baseline performance, the active learning Koopman operator model continuously improves tracking performance as the experiment progresses.

In Fig. 4 we show the experimental results resulting from the active and passive Koopman operator learning approaches. The online learning approach is much more effective at tracking the desired trajectory than the fixed learning approach. Moreover, we notice that as the experiment progresses, the active Koopman operator trajectories continue to improve. The Koopman operator synthesized through active learning required no *a priori* knowledge of the system dynamics, bootstrapped initial model guess, or specification of granular media physics. Despite the fact that this problem could have been solved by other means, this experimental application is facilitated by the Koopman operator model representation.

4.2 Rapid Quadcopter Stabilization with Active Learning

The Koopman operator's ability to actively acquire information in real-time allows us to approach problems that demand rapid reactive efforts. For settings in which large disturbances permanently affect system dynamics, we must be able to quickly develop new models to adapt to the disturbance while still achieving an underlying goal. We demonstrate the Koopman operator's quick adaptability by trying to a stabilize a quadcopter with a malfunctioning motor in simulation. We define the quadcopter malfunction as a single motor operating at 80% capacity. However, the malfunction is unmodeled and unknown to the Koopman operator ahead of time, which means it must be learned in real time in order to stabilize the quadcopter and prevent it from falling, as shown in [1]. The quadcopter configuration is shown in

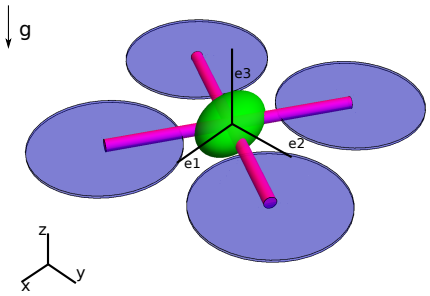


Fig. 5 Quadcopter vehicle model configuration in body-fixed frame for active learning of vehicle dynamics under emergency conditions. The quadcopter dynamics are defined such that one motor has an unspecified malfunction and can only work at 80% capacity.

Fig. 5, and the system dynamics used to carry out this simulated experiment are

$$\begin{aligned} \dot{h} &= h \begin{bmatrix} \hat{\omega} & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix} \\ J\dot{\omega} &= M + J\omega \times \omega \\ \dot{\mathbf{v}} &= \frac{1}{m} F e_3 - \omega \times \mathbf{v} - g R^T e_3, \end{aligned} \tag{44}$$

where the dynamics are defined on the Lie group $h = (R, p) \in SE(3)$. The control inputs to the system are $\mathbf{u} = [u_1, u_2, u_3, u_4]$ and

$$\begin{aligned} F &= k_t(u_1^2 + u_2^2 + u_3^2 + u_4^2) \\ M &= \begin{bmatrix} k_t l(u_2^2 - u_4^2) \\ k_t l(u_3^2 - u_1^2) \\ k_m(u_1^2 - u_2^2 + u_3^2 - u_4^2) \end{bmatrix}. \end{aligned} \tag{45}$$

Details regarding implementation of the quadcopter dynamics are covered in [13]. The set of basis functions chosen directly embed the quadcopter dynamics, such that the Koopman operator has a bootstrapped model of the nominal system dynamics

$$\begin{aligned} \Psi_x(\mathbf{x}) &= [\mathbf{a_g}, \omega, \mathbf{v}, g(\mathbf{v}, \omega)]^T \in \mathbb{R}^{18} \\ \Psi_u(\mathbf{u}) &= \mathbf{u} \in \mathbb{R}^4, \end{aligned} \tag{46}$$

where $\mathbf{a_g}, \mathbf{v}, \omega \in \mathbb{R}^3$, $\mathbf{a_g}$ is the body-centered gravity vector, $\mathbf{v}$ is a linear velocity vector, and ω is the angular velocity vector. The additional basis functions $g(\mathbf{v}, \omega)$ are linear combinations of the velocity vectors, and were chosen without any *a priori* knowledge of the motor malfunction specification

$$g(\mathbf{v}, \omega) = [v_1\omega_1, v_1\omega_2, v_1\omega_3, v_2\omega_1, v_2\omega_2, v_2\omega_3, v_3\omega_1, v_3\omega_2, v_3\omega_3].$$

The experimental procedure consists of setting the quadcopter in free-fall from

some initial condition and for a period of 1s learning a Koopman operator online with respect to a specified objective. Active learning trials applied an information maximizing objective during the learning period, then switched to a stabilizing objective afterward. Passive learning trials followed a stabilizing objective for the entirety of each trial. After the 1s learning period we did not continue to update the operator online for either set of trials. A set of 100 Monte-Carlo trials over uniformly distributed initial conditions were carried out. Figure 6 summarizes the experimental results. Figure 6A depicts timestamps from a pair of quadcopter trajectories under active and passive learning.

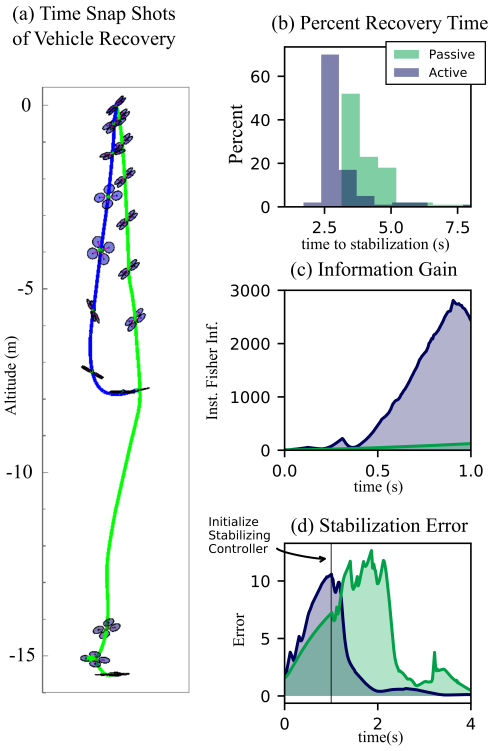


Fig. 6 Quadcopter vehicle free-fall simulation where one motor has an unmodeled malfunction. The goal is to learn to stabilize the quadcopter under two learning approaches. The first is a passive learning approach (green) where a Koopman operator is learned online over 1s while attempting to stabilize the quadcopter. The second is an active learning approach (blue) where a Koopman operator is generated while maximizing the Fisher information metric over 1s, then the system switches objectives to achieve stabilization. The active learning approach outperforms the passive approach.

In Fig. 6B we display the results from the Monte-Carlo trials. The active Koopman operator stabilized the quadcopter within 2.5s in over 65% of trials, while the passive operator took another full second on average. Figure 6C shows the instantaneous information gain over the 1s learning period for both passive and active learning approaches, where the active learning approach acquires orders of magnitude more information than the passive approach in a selected trial. Finally, Fig. 6D depicts the stabilization error for a particular trial with passive and active learning. The active learning trial stabilizes the quadcopter more quickly than the passive learning approach and with lower error after the stabilizing objective switch.

Through this experiment, we have shown that the Koopman operator with active learning can be used in time-critical applications. We introduced an unmodeled motor malfunction into the quadcopter dynamics and learned a Koopman operator in real time in order to stabilize the system. We showed that by first executing information maximizing trajectories prior to attempting to stabilize we outperform passive online learning techniques.

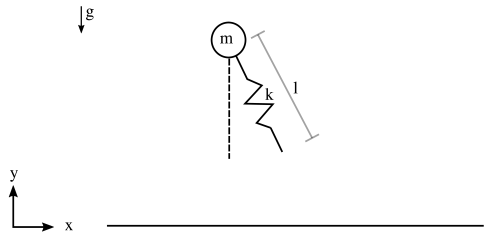


Fig. 7 SLIP model configuration. The SLIP system obeys a set of hybrid dynamics corresponding to the separate stance and flight modes.

4.3 Learning SLIP Hybrid Dynamics

Hybrid systems can be difficult to model because of inherent discontinuities in the system dynamics. These discontinuities often cannot be represented by a single model. Experimental learning in uncertain environments in the real world may demand learning techniques that model such discontinuities. In particular, field robotics often concerns itself with modeling locomotive systems, such as bipedal or quadrupedal walkers. Walking robots deal with the discontinuities of impact at each step of their locomotive cycle, and as such must have a way to reason about them. The simplest dynamical walker model is the Spring-Loaded Inverted Pendulum (SLIP) model, shown in Fig. 7.

The SLIP system dynamics are split into two hybrid modes: stance and flight. The transition between these hybrid modes is described by an indicator function, often referred to as a guard equation, which specifies the set of dynamics evolving the system at a given coordinate in state-space. The system states are $\mathbf{x} = [x, y, \dot{x}, \dot{y}, x_f]^T$, where (x, y) indicates the position of the mass in the configuration specified in Fig. 7, and x_f is the position of the SLIP model's foot. The model's control inputs are $\mathbf{u} = [u_s, u_f]^T$. The dynamics of the SLIP model are

$$f_{stance} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ (k(l_0 - l(x)) + u_s)\frac{x - x_f}{ml(x)} \\ (k(l_0 - l(x)) + u_s)\frac{y}{ml(x)} - g \\ 0 \end{bmatrix}, \quad f_{flight} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \\ -g \\ \dot{x} + u_f \end{bmatrix}, \tag{47}$$

where l_0 is the resting length of the spring, and $l(x) = \sqrt{(x - x_f)^2 + y^2}$ [20]. Then, the indicator function is $\Phi_{SLIP}(x) = \text{sign}(1 - \frac{l_0}{l(x)}) \in \{-1, 1\}$. The complete hybrid dynamics are

$$f_{SLIP}(x, u) = \begin{cases} f_{stance}(x, u), & \text{if } \Phi_{SLIP}(x) = -1 \\ f_{flight}(x, u), & \text{otherwise} \end{cases}. \tag{48}$$

We use the hybrid formulation of the Koopman operator as specified in Section 2.5, as well as the analytical indicator function to actively learn a hybrid Koopman operator representation of the SLIP model. For the first $10s$ we have a simulated

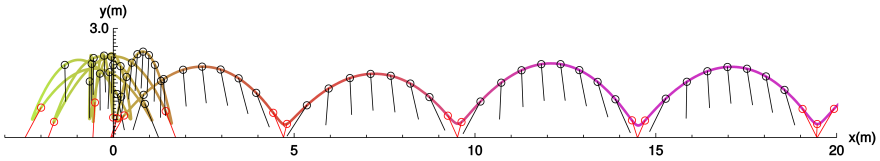


Fig. 8 Active identification of simulated SLIP system dynamics. The SLIP system first obeys an information maximizing objective for $10s$, and then the objective is changed to following a forward velocity trajectory.

SLIP hopper follow an information maximizing trajectory, where each hybrid Koopman mode incrementally updates according to the indicator function. Following the active learning period, the objective changes and the model is made to follow a forward hopping velocity of $4.2m/s$. Results are shown in Fig. 8 where the hybrid Koopman model first maximizes information about its dynamics for $10s$, and then it moves forward tracking a trajectory. Figure 9 depicts the SLIP hopper's switch in objectives in terms of the model's forward velocity.

The Koopman operator's hybrid formulation enables modeling of discontinuous dynamical systems, which is of great importance in the field of locomotion, but also field robotics at large. Furthermore, the experiment presented an active learning framework for characterizing hybrid dynamical systems with known guard equations that could be extended to other walking robots.

4.4 Generating Finite Automata via DSS

In Section 4.3 we discussed the importance of modeling hybrid system dynamics in experimental field robotics. Although we were able to generate a hybrid Koopman model for the SLIP hopper, we required *a priori* specification of the system's indicator function. The indicator function is not generally known ahead of time. Thus, it is of interest to develop a systematic approach to identifying hybrid system dynamics, as well as the underlying indicator function specifying state-space boundaries between hybrid modes.

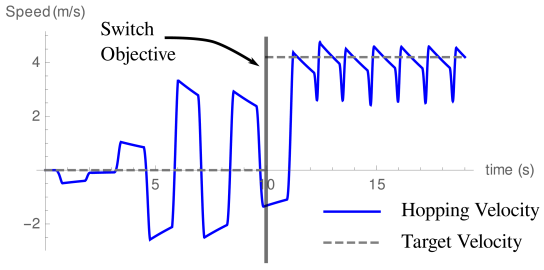


Fig. 9 SLIP model objectives. First, the SLIP hopper follows an information maximizing objective for 10s while learning a Koopman representation of its dynamics. Then, the SLIP hopper uses this model to carry out a forward trajectory.

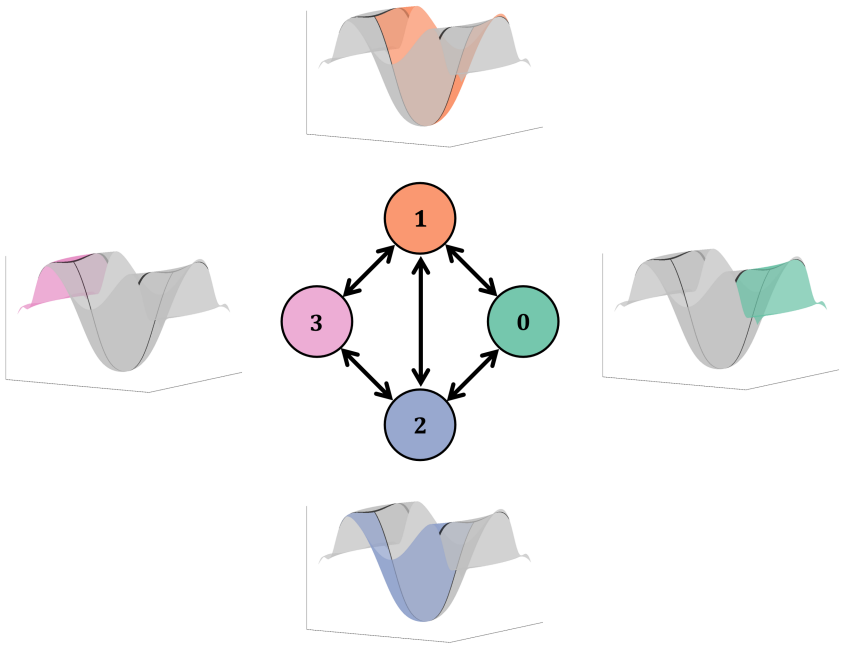


Fig. 10 Segmentation of a non-hybrid dynamical system into a hybrid automaton represented as a graph. Each node in the graph is a distinct dynamical system represented by a Koopman operator governing the system dynamics over a region of the underlying state-space manifold. The partitions of the manifold are specified by an SVM model $\Phi(\Psi(x))$. Figure shown in [4].

Hybrid dynamical systems are often represented as finite automata with discrete and continuous components. While the structure of the hybrid automaton is discrete, each of its nodes is a continuous dynamical system. The transitions are specified by the indicator function and guard equations. This graphical model is a compact representation of the system dynamics [18]. Dynamical System Segmentation (DSS), proposed in [4], is an algorithm that generates state-space partitions of dynamical systems from data, thereby generating a hybrid automaton. For hybrid systems, DSS can identify the dynamics of each hybrid mode and generate an indicator function from data. For non-hybrid systems, DSS synthesizes a finite automaton representation of the dynamics from data. The state-space continuous dynamics are partitioned into a set of distinct dynamics governing a given region of the state-space manifold.

Given a dataset $X = [x_1, ..., x_M]$ from some dynamical system and a set of basis functions $\Psi(x) \in \mathbb{R}^N$, DSS synthesizes a set of Koopman operators from subsets of the dataset $K_i = \mathit{GenerateKoopman}(X_{a:b}),\ s.t.\ X_{a:b} = [x_a, ..., x_b]$. The choice of method for splitting the dataset X into subsets is left to the user. This set of W Koopman operators $\mathcal{K} = \{K_1, ..., K_W\}$ are each a local estimate of the system dynamics for some neighborhood of the state-space manifold. However, given that this set $\mathcal{K}$ may have redundancies we are interested in distilling a minimal set of Koopman operators to represent the system dynamics. By considering each Koopman operator

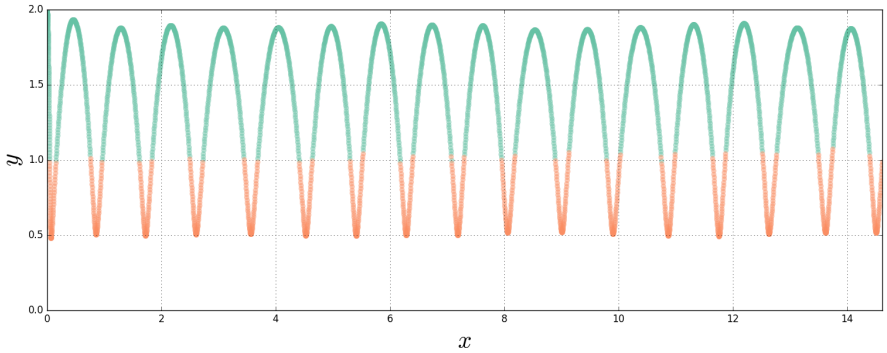


Fig. 11 Dynamical system segmentation of the SLIP model system described in Section 4.3. The DSS algorithm without any *a priori* knowledge of the SLIP system dynamics is able to discern the two hybrid modes of flight and stance, as well as the indicator function that switches between modes. The plot shows the position of the SLIP hopper's mass color coded according to the hybrid mode it is currently in. Green corresponding to flight, orange to stance.

$K_i \in \mathbb{R}^{N\times N}$ as points in $\mathbb{R}^{N^2}$ space, we can use a clustering algorithm to derive exemplar operators from the set $\mathcal{K}$. We suggest non-parametric clustering algorithms, such as HDBSCAN [8], in order to avoid presupposing a number of clusters. We will consider the minimal set of operators $\overline{\mathcal{K}}$ as a node set in a finite automaton. To derive the edge set $\mathcal{E} = \{K_1, ..., K_B\}$, we generate an SVM $\Phi(\Psi(x))$ using the labeled dataset X and determine state-space transitions between nodes [28]. This graphical model $\mathbb{G} = (\overline{\mathcal{K}}, \mathcal{E})$ is the output of DSS. Figure 10 depicts a notional example of DSS applied to some dynamical system, as well as the relationship between the graph and the partitioned state-space manifold.

To demonstrate the DSS algorithm's performance in identifying hybrid systems we applied DSS to a dataset of a SLIP hopper as formulated in Section 4.3 following a forward constant velocity trajectory. Figure 11 depicts a successful identification of the bimodal SLIP dynamics purely from data without *any* prior knowledge of the system dynamics. The data-driven SVM indicator function $\Phi(\Psi(x))$ correctly captures the SLIP system transitions between flight and stance modes.

Additionally, we segment non-hybrid systems to showcase the resulting finite descriptions of state-space continuous systems. We use the well-studied cart-pendulum

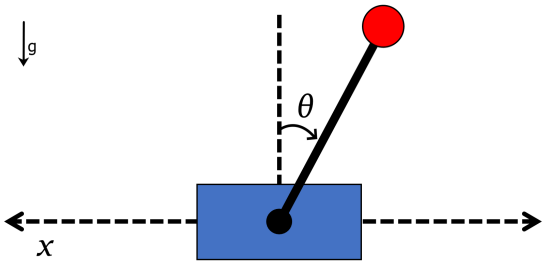


Fig. 12 Cart-pendulum system used for DSS, with its unstable equilibrium defined at $\theta = 0$. Figure shown in [4].

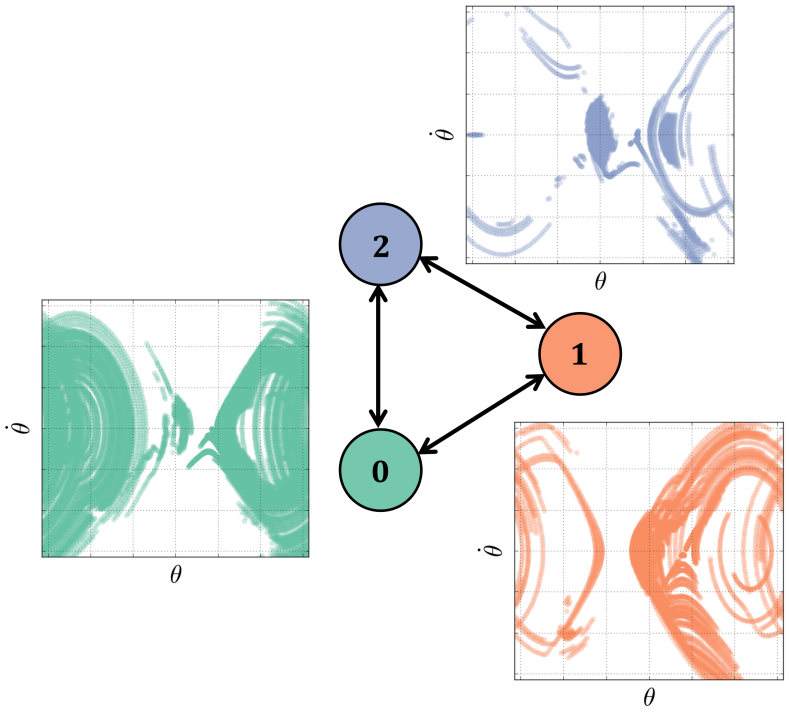


Fig. 13 Application of DSS to a set of 30 control solutions to the cart-pendulum inversion problem. Each node in the generated graph represents a Koopman operator that describes the cart-pendulum nonlinear dynamics over a particular region of the state-space manifold. Mode 0 corresponds to energy pumping and swing-up, mode 1 corresponds to energy removal and slow-down, and mode 2 corresponds to stabilization. Figure shown in [4].

inversion problem as an example. The cart-pendulum system, shown in Fig 12, is actuated about its horizontal axis and has $\theta = 0$ defined at the unstable equilibrium point. We used the same nonlinear model-predictive controller as in Section 4.1, SAC, to synthesize a set of 30 trajectories with the goal of inverting and stabilizing the cart-pendulum system. We then applied DSS to this dataset and found that the pendulum inversion task could be segmented into three hybrid modes corresponding to swing-up, slow-down, stabilization dynamics. The synthesized finite automaton resulting from DSS is shown in Fig 13. Each node is accompanied by its corresponding partition of the original dataset shown in the $(\theta, \dot{\theta})$ phase plane.

Dynamical system segmentation is an extension of the hybrid dynamical formulation of the Koopman operator described in Section 2.5, and showcased in Section 4.3. DSS generates finite representations of dynamical systems from data without any prior knowledge of the system. As an experimental tool, as was shown in [4], DSS is capable of discerning complex relationships from motion data and formulating them in a low-dimensional representation. Identification of discontinuous phenomena from data is of great importance in experimental settings where environ-

mental interactions are often difficult to model, and the Koopman operator provides us with the ability to formulate algorithms to address this problem.

5 Conclusion

Experimental learning is necessary for real-time contexts where the environment and internal dynamics can change rapidly. The learning techniques we apply should reflect that these settings demand incremental learning in time-varying environments, and most machine learning techniques do not. We have proposed the EDMD-synthesized finite-dimensional Koopman operator as a technique well-suited to experimental settings and compared it to similar techniques to characterize their relationships. Moreover, we provide a set of conditions that specify the relationship between EDMD and alternative learning methods. We highlight the synthesized Koopman operator's capabilities in a variety of experimental applications. Modeling real processes requires techniques capable of purposeful learning, and we propose the Koopman operator as a suitable model representation for these settings.

Acknowledgements This work was supported by the National Science Foundation under grant CBET-1637764. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Abraham, I., Murphey, T.: Active learning of dynamics and data-driven control. IEEE Transactions on Robotics (Submitted)
2. Abraham, I., Torre, G., Murphey, T.: Model-based control using Koopman operators. Robotics: Science and Systems (2017)
3. Ansari, A., Murphey, T.D.: Sequential action control: Closed-form optimal control for nonlinear and nonsmooth systems. IEEE Transactions on Robotics **32** (2017)
4. Berrueta, T.A., Pervan, A., Fitzsimons, K., Murphey, T.: Dynamical system segmentation for information measures in motion. IEEE Robotics and Automation Letters (Submitted)
5. Bishop, C.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer (2006)
6. Brunton, S., Brunton, B., Proctor, J., Kutz, J.: Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. PLOS ONE **11**(2), 1–19 (2016)
7. Budišić, M., Mohr, R., Mezić, I.: Applied Koopmanism. Chaos: An Interdisciplinary Journal of Nonlinear Science **22**(4), 047,510 (2012)
8. Campello, R., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Advances in Knowledge Discovery and Data Mining, pp. 160–172. Springer (2013)
9. Chen, B.W., Abdullah, N.N.B., Park, S., Gu, Y.: Efficient multiple incremental computation for kernel ridge regression with Bayesian uncertainty modeling. Future Generation Computer Systems **82**, 679 – 688 (2018)
10. Chirikjan, G.S.: Stochastic Models, Information Theory, and Lie Groups, Volume 2. Springer (2012)

11. Dias, J., Leitao, J.: A method for computing the information matrix of stationary Gaussian processes. In: 1996 8th European Signal Processing Conference (EUSIPCO 1996), pp. 1–4 (1996)
12. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. IEEE Transactions on Signal Processing **52**(8), 2275–2285 (2004)
13. Fan, T., Murphey, T.: Online feedback control for input-saturated robotic systems on lie groups. Robotics: Science and Systems (2016)
14. Gallager, R.: Stochastic Processes: Theory for Applications. Cambridge University Press (2013)
15. Govindarajan, N., Arbabi, H., van Blargian, L., Matchen, T., Tegling, E., Mezić, I.: An operator-theoretic viewpoint to non-smooth dynamical systems: Koopman analysis of a hybrid pendulum. In: 2016 IEEE 55th Conference on Decision and Control (CDC), pp. 6477–6484 (2016)
16. Hamilton, J.: Time Series Analysis. Princeton University Press (1994)
17. Heersink, B., Warren, M., Hoffmann, H.: Dynamic mode decomposition for interconnected control systems (2017). DOI abs/1709.02883
18. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–283 (1996)
19. Kaiser, E., Kutz, J.N., Brunton, S.L.: Data-driven discovery of Koopman eigenfunctions for control. In: arXiv preprint (2017). DOI arXiv:1707.01146
20. Kalinowska, A., Fitzsimons, K., Dewald, J.P., Murphey, T.D.: Online user assessment for minimal intervention during task-based robotic assistance. In: Robotics: Science and Systems (2018)
21. Koopman, B.: Hamiltonian systems and transformation in Hilbert space. Proceedings of the National Academy of Sciences **17**(5), 315–318 (1931)
22. Korda, M., Mezić, I.: Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. Automatica **93**, 149 – 160 (2018)
23. Lee, J., Bahri, Y., Novak, R., Schoenholz, S.S., Pennington, J., Sohl-Dickstein, J.: Deep neural networks as Gaussian processes. In: arXiv preprint (2018). DOI arXiv:1711.00165
24. Mezić, I.: Analysis of fluid flows via spectral properties of the Koopman operator. Annual Review of Fluid Mechanics **45**(1), 357–378 (2013). DOI 10.1146/annurev-fluid-011212-140652
25. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)
26. Neal, R.M.: Priors for infinite networks. In: Bayesian Learning for Neural Networks, pp. 29–53. Springer (1996)
27. Nguyen-Tuong, D., Peters, J., Seeger, M.: Local Gaussian process regression for real time online model learning and control. In: Proceedings of the 21st International Conference on Neural Information Processing Systems, pp. 1193–1200 (2008)
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
29. Peitz, S., Klus, S.: Koopman operator-based model reduction for switched-system control of PDEs. In: arXiv preprint (2017). DOI arXiv:1710.06759
30. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press (2005)
31. Schmid, P.J.: Dynamic mode decomposition of numerical and experimental data. Journal of Fluid Mechanics **656**, 5–28 (2010)
32. Seal, H.L.: Studies in the history of probability and statistics. XV: The historical development of the Gauss linear model. Biometrika **54**(1/2), 1–24 (1967)
33. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016)
34. Srivastava, J., Anderson, D.: A comparison of the determinant, maximum root, and trace optimality criteria. Communications in Statistics **3**(10), 933–940 (1974)

35. Tu, J., Rowley, C., Luchtenburg, D., Brunton, S., Kutz, J.: On dynamic mode decomposition: Theory and applications. Journal of Computational Dynamics **1**, 391 (2014). DOI 10.3934/jcd.2014.1.391
36. Umlauft, J., Beckers, T., Kimmel, M., Hirche, S.: Feedback linearization using Gaussian processes. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pp. 5249–5255 (2017)
37. Van Vaerenbergh, S., Fernandez-Bes, J., Elvira, V.: On the relationship between online Gaussian process regression and kernel least mean squares algorithms. In: 2016 IEEE International Workshop on Machine Learning for Signal Processing (2016)
38. Williams, M., Kevrekidis, I., Rowley, C.: A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition. Journal of Nonlinear Science **25**, 1307–1346 (2015)