Data-Driven Control of Soft Robots Using Koopman Operator Theory

Daniel Bruder, Xun Fu, R. Brent Gillespie, C. David Remy, and Ram Vasudevan

***Abstract*—Controlling soft robots with precision is a challenge due to the difficulty of constructing models that are amenable to model-based control design techniques. Koopman operator theory offers a way to construct explicit dynamical models of soft robots and to control them using established model-based control methods. This approach is data driven, yet yields an explicit control-oriented model rather than just a "black-box" input–output mapping. This work describes a Koopman-based system identification method and its application to model predictive control (MPC) design for soft robots. Three MPC controllers are developed for a pneumatic soft robot arm via the Koopman-based approach, and their performances are evaluated with respect to several real-world trajectory following tasks. In terms of average tracking error, these Koopman-based controllers are more than three times more accurate than a benchmark MPC controller based on a linear state-space model of the same system, demonstrating the utility of the Koopman approach in controlling real soft robots.**

***Index Terms*—Koopman operator, model learning for control, optimal control, soft robots.**

I. Introduction

SOFT robots have bodies made of intrinsically soft and/or compliant materials. This softness enables them to safely interact with delicate objects and to passively adapt their shape to unstructured environments [1]. Such traits are desirable for robotic applications that demand safe human–robot interaction, such as wearable robots, in-home assistive robots, and medical robots. Unfortunately, the soft bodies of these robots also impose modeling and control challenges, which have restricted their functionality to date. While many novel soft devices such as soft grippers [2], crawlers [3], and swimmers [4] exploit the flexibility of their bodies to achieve coarse behaviors such as grasping and locomotion, they do not exhibit precise control capabilities.

The challenge of achieving precise control is largely due to the difficulty of devising models of soft robots that are amenable to model-based control design techniques. Consider, for instance, a rigid-bodied robotic system that is made up of rigid links connected together by joints. Since joint displacements can be used to fully describe the configuration of a rigid-bodied system, joint displacements and their derivatives make a natural choice for the state variables for rigid-bodied robots [5]. One can use this choice of state variables to describe the dynamics of the rigid-bodied robot. This, as a result, makes the application of model-based control design techniques such as feedback linearization [5], nonlinear model predictive control [6], linear–quadratic regulator trees [7], sequential action control [8], and others feasible.

Soft robots, in contrast, do not exhibit localized deformation at joints, but instead deform continuously along their bodies and have infinite degrees of freedom. In the absence of joints, there does not exist a canonical choice of state variables to describe the geometry of a soft robot. As a result, existing representations are typically only rich enough to describe the system under simplifying physical assumptions. For example, the popular piecewise constant curvature model [9] provides a low-dimensional description of the shape of continuum robots, but only under the assumption that their shapes can be represented by consecutive arcs. Other simplified models such as pseudo-rigid-body [10], quasi-static [11]–[14], or simplified geometry [15]–[18] have proven useful, but they are only able to describe behavior in the subset of conditions over which their physical simplifying assumptions hold. This makes these models insufficient for model-based control in tasks that would require violating their physical simplifying assumptions. For example, a controller based on the soft actuator model presented in [16] would be insufficient for tasks that require bending, since the model assumes cylindrical geometry of the actuator.

Alternatively, data-driven methods such as traditional machine learning and deep learning can be applied to construct models without making such physical simplifying assumptions.

Manuscript received August 5, 2020; accepted October 13, 2020. This work was supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant 1256260 DGE, in part by the Naval Research under Grant N00014-18-1-2575, and in part by the National Science Foundation under Grant 1751093. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Toyota Research Institute (TRI) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. This article was recommended for publication by Associate Editor J. Kober and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Daniel Bruder.)*

Daniel Bruder was with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109 USA. He is now with the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: dbruder@seas.harvard.edu).

Xun Fu, R. Brent Gillespie, and Ram Vasudevan are with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: xunfu@umich.edu; brentg@umich.edu; ramv@umich.edu).

C. David Remy is with the Institute for Nonlinear Mechanics, University of Stuttgart, 70174 Stuttgart, Germany (e-mail: remy@inm.uni-stuttgart.de).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2020.3038693. The material consists of a video. The video introduces the hardware used in experiments and shows the robotic system performing various trajectory-following tasks using the control algorithms described in this article. Contact dbruder@seas.harvard.edu for further questions about this work.

Digital Object Identifier 10.1109/TRO.2020.3038693

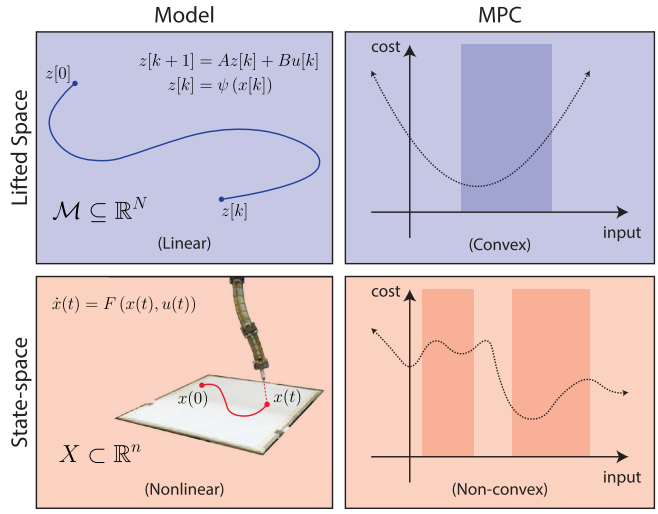


Fig. 1. Nonlinear dynamical system (bottom-left) has a *linear* representation in the *lifted* space made up of all real-valued functions (top-left). While a model predictive controller designed for the nonlinear system in state space requires solving a nonconvex optimization problem to choose inputs at each time step (bottom-right), this problem is convex for a model predictive controller designed for the lifted linear system (top-right). This article develops a data-driven method to construct such a lifted model representation for soft robotic systems in the presence of outliers as well as a convex-model-based controller for such systems.

These models provide a "black-box" mapping from inputs to outputs and have been shown to predict behavior well across various configurations of soft robots [12], [19]. However, the structure of these types of models is not amenable to existing model-based control design techniques, since they are typically nonlinear and may not be easily invertible.

Koopman operator theory offers a data-driven approach that avoids physical simplifying assumptions, but also yields explicit control-oriented models. The approach leverages the linear structure of the Koopman operator to construct linear models of nonlinear controlled dynamical systems from input–output data [20]–[24], and to control them using established linear control methods [25]–[28]. In theory, this approach involves *lifting* the state space to an infinite-dimensional space of scalar functions (referred to as observables), where the flow of such observables along trajectories of the nonlinear dynamical system is described by the *linear* Koopman operator. In practice, however, it is infeasible to compute an infinite-dimensional operator, so a process called extended dynamic mode decomposition (EDMD) [29] is typically employed to compute a finite-dimensional projection of the Koopman operator onto a finite-dimensional subspace of all observables (scalar functions). This approximation of the Koopman operator describes the evolution of the output variables themselves, provided that they lie within the finite subspace of observables upon which the operator is projected. Hence, this approach makes it possible to control the output of a nonlinear dynamical system using a controller designed for its linear Koopman representation (see Fig. 1).

The Koopman approach to modeling and control is well suited for soft robots for several reasons. Soft robots pose less of a physical threat to themselves or their surroundings when subjected to random control inputs than conventional rigid-bodied robots. This makes it possible to safely collect input–output data over a wide range of operating conditions, and to do so in an automated fashion. Furthermore, since the Koopman procedure is entirely data driven, it inherently captures input–output behavior and avoids the ambiguity involved in choosing a discrete set of states for a structure with infinite degrees of freedom.

Portions of the work presented here originally appeared in [30]. That work introduced a modification to the Koopman *linear* system identification procedure described in [26] to make the resulting Koopman operator both more sparse and less sensitive to outliers and noise in the training data. It then applied that approach to model and control a physical soft robot arm using linear model predictive control (MPC). This work builds on those previous contributions by incorporating the Koopman-based *nonlinear* system identification procedure described in [21] and using it to control a physical soft robot using nonlinear model predictive control (NMPC). It includes a comparison between these Koopman-based controllers, which highlights the relative strengths and weaknesses of linear and nonlinear model predictive controlllers applied to soft robots.

The rest of this article is organized as follows. In Section II, we formally introduce the Koopman operator and describe how it is used to construct models of nonlinear dynamical systems from data. In Section III, we describe how the Koopman model can be used for linear model predictive control (MPC) and nonlinear model predictive control (NMPC). In Section IV, we describe the set of experiments used to evaluate the performance of Koopman-based model predictive controllers on a real soft robot platform. In Section V, the results of these experiments are discussed and interpreted. Finally, Section VI concludes this article.

II. System Identification

Any finite-dimensional Lipschitz continuous nonlinear dynamical system has an equivalent infinite-dimensional linear representation in the space of all scalar-valued functions of the system's state [31, Def. 3.3.1]. This linear representation, which is called the Koopman operator, describes the flow of functions along trajectories of the system. While it is not possible to numerically represent the infinite-dimensional Koopman operator, it is possible to represent its projection onto a finite-dimensional subspace as a matrix. This section shows that for a given choice of basis functions, a dynamical system model can be extracted directly from such a matrix approximation of the Koopman operator.

The remainder of this section outlines an approach for approximating the Koopman operator from data and using it to construct a linear or nonlinear system model. Section II-A introduces the Koopman operator. Section II-B describes a process by which a matrix approximation of the Koopman operator can be fit from data. Sections II-C and II-D present the construction of a linear/nonlinear dynamical model using a matrix approximation of the Koopman operator. Section II-E presents modifications to the system identification process to promote sparsity and reduce overfitting of the resulting models.

A. Koopman Representation of a Dynamical System

Consider a dynamical system

$$\dot{x}(t) = F(x(t)) \tag{1}$$

where $x(t) \in X \subset \mathbb{R}^n$ is the state of the system at time $t \geq 0$, X is a compact subset, and F is a continuously differentiable function. Denote by $\phi_t(x_0)$ the solution to (1) at time t when beginning with the initial condition x_0 at time 0. ϕ_t is referred to as the *flow map*.

The system can be lifted to an infinite-dimensional function space $\mathcal{F}$ composed of all continuous real-valued functions with compact domain $X \subset \mathbb{R}^n$. Elements of $\mathcal{F}$ are called *observables*. In $\mathcal{F}$, the flow of the system is characterized by the set of Koopman operators $U_t : \mathcal{F} \to \mathcal{F}$, for each $t \geq 0$, which describes the evolution of the observables $f \in \mathcal{F}$ along the trajectories of the system according to the following definition:

$$U_t f = f \circ \phi_t \tag{2}$$

where $\circ$ indicates function composition. As desired, U_t is a linear operator even if the system (1) is nonlinear, since for $f_1, f_2 \in \mathcal{F}$ and $\lambda_1, \lambda_2 \in \mathbb{R}$

$$\begin{aligned} U_t(\lambda_1 f_1 + \lambda_2 f_2) &= \lambda_1 f_1 \circ \phi_t + \lambda_2 f_2 \circ \phi_t \\ &= \lambda_1 U_t f_1 + \lambda_2 U_t f_2. \end{aligned} \tag{3}$$

Thus, the Koopman operator provides a linear representation of the flow of a nonlinear system in the infinite-dimensional space of observables [32]. Contrast this representation with the one generated by the (nonlinear) flow map that for each $t \geq 0$ describes how the initial condition evolves according to the dynamics of the system. In particular, if one wants to understand the evolution of an initial condition x_0 at time t according to (1), then one could solve the nonlinear differential equation to generate the flow map. In contrast, one could apply U_t (a linear operator) to the function that projects onto each component of the state to generate the flow map for each component of the state.

B. Identification of the Koopman Operator

Since the Koopman operator is an infinite-dimensional object, it cannot be represented by a finite-dimensional matrix. Therefore, we settle for the projection of the Koopman operator onto a finite-dimensional subspace. Using a modified version of the EDMD algorithm [21], [22], [29], we identify a finite-dimensional approximation of the Koopman operator via linear regression applied to observed data. The remainder of this subsection describes the mathematical underpinnings of this process.

Define $\bar{\mathcal{F}} \subset \mathcal{F}$ to be the subspace of $\mathcal{F}$ spanned by $N > n$ linearly independent basis functions $\{\psi_i : \mathbb{R}^n \to \mathbb{R}\}_{i=1}^N$. We denote the image of ψ_i as $\mathcal{R}_i$, which is equal to $\{w \in \mathbb{R} | \exists x \in \mathbb{R}^n, \psi_i(x) = w\}$. For convenience, we assume that the first n basis functions are defined as

$$\psi_i(x) = x_i \tag{4}$$

where x_i denotes the ith element of x. Any observable $\bar{f} \in \bar{\mathcal{F}}$ can be expressed as a linear combination of elements of these basis functions

$$\bar{f} = \theta_1 \psi_1 + \cdots + \theta_N \psi_N \tag{5}$$

where each $\theta_i \in \mathbb{R}$. To aid in presentation, we define the vector of coefficients $\theta := [\theta_1 \ \cdots \ \theta_N]^\top$ and define the *lifting function* $\psi : \mathbb{R}^n \to \mathbb{R}^N$ as

$$\psi(x) := \begin{bmatrix} x_i & \cdots & x_n & \psi_{n+1}(x) & \cdots & \psi_N(x) \end{bmatrix}^\top. \tag{6}$$

We denote the image of ψ as $\mathcal{M} = \mathcal{R}_1 \times \cdots \times \mathcal{R}_N \subset \mathbb{R}^N$. By (5) and (6), $\bar{f}$ evaluated at a point x in the state space is given by

$$\bar{f}(x) = \theta^\top \psi(x). \tag{7}$$

We, therefore, refer to $\psi(x)$ as the *lifted state*, and θ as the *vector representation* of $\bar{f}$.

Given this vector representation for observables, a linear operator on $\bar{\mathcal{F}}$ can be represented as an $N \times N$ matrix. We denote by $\bar{U}_t \in \mathbb{R}^{N \times N}$ the approximation of the Koopman operator on $\bar{\mathcal{F}}$, which operates on observables via matrix multiplication

$$\bar{U}_t \theta = \theta' \tag{8}$$

where θ and θ' are each vector representations of observables in $\bar{\mathcal{F}}$. Our goal is to find a $\bar{U}_t$ that describes the action of the infinite-dimensional Koopman operator U_t as accurately as possible in the L^2-norm sense on the finite-dimensional subspace $\bar{\mathcal{F}}$ of all observables.

To perfectly mimic the action of U_t on any observable $\bar{f} \in \bar{\mathcal{F}} \subset \mathcal{F}$, according to (2), the following should be true for all $x \in X$:

$$\bar{U}_t \bar{f}(x) = \bar{f} \circ \phi_t(x) \tag{9}$$

$$(\bar{U}_t \theta)^\top \psi(x) = \theta^\top \psi \circ \phi_t(x) \tag{10}$$

$$\bar{U}_t^\top \psi(x) = \psi \circ \phi_t(x) \tag{11}$$

where (10) follows by substituting (7), and (11) follows since the result holds for all $\bar{f}$. Since this is a linear equation, it follows that for a given $x \in X$, solving (11) for $\bar{U}_t$ yields the best approximation of U_t on $\bar{\mathcal{F}}$ in the L^2-norm sense [33]:

$$\bar{U}_t = \left(\psi^\top(x)\right)^\dagger \left(\psi \circ \phi_t(x)\right)^\top \tag{12}$$

where superscript $\dagger$ denotes the Moore–Penrose pseudoinverse.

To approximate the Koopman operator from a set of experimental data, we take K discrete state measurements in the form of so-called snapshot pairs $(a[k], b[k])$ for each $k \in \{1, \ldots, K\}$, where

$$a[k] = x[k] \tag{13}$$

$$b[k] = \phi_{T_s}(x[k]) + \sigma[k] \tag{14}$$

where $\sigma[k]$ denotes the measurement noise, T_s is the sampling period which is assumed to be identical for all snapshot pairs, and $x[k]$ denotes the measured state corresponding to the kth measurement. Note that consecutive snapshot pairs do not have to be generated by consecutive state measurements. We then lift all of the snapshot pairs according to (6) and compile them into

the following $K \times N$ matrices:

$$\Psi_a := \begin{bmatrix} \psi(a[1])^\top \\ \vdots \\ \psi(a[K])^\top \end{bmatrix} \qquad \Psi_b := \begin{bmatrix} \psi(b[1])^\top \\ \vdots \\ \psi(b[K])^\top \end{bmatrix}. \tag{15}$$

$\bar{U}_{T_s}$ is chosen so that it yields the least-squares best fit to all of the observed data, which, following from (12), is given by

$$\bar{U}_{T_s} := \Psi_a^\dagger \Psi_b. \tag{16}$$

The dynamics of mechanical systems are described by second-order differential equations. Expressed in state-space form, the state of such a system includes both the positions and velocities of a set of generalized coordinates. Therefore, when identifying the dynamics of a mechanical system from data, it is prudent to include velocities in the state. For a discrete system representation, velocity information can be included by incorporating a delay into the set of snapshot pairs, since it encodes the change in the value of the measured state over a single time step. To incorporate one or more delays, we define the snapshot pairs as

$$a[k] = \begin{bmatrix} x[k]^\top, & x[k-1]^\top & \dots, & x[k-d]^\top \end{bmatrix}^\top \tag{17}$$

$$b[k] = \begin{bmatrix} (\phi_{T_s}(x[k]) + \sigma_k)^\top & x[k]^\top & \dots & x[k-d+1]^\top \end{bmatrix}^\top \tag{18}$$

where d is the number of delays. We then modify the domain of the lifting function such that $\psi : \mathbb{R}^{n+nd} \to \mathbb{R}^N$ to accommodate the larger dimension of the snapshot pairs. Once these snapshot pairs have been assembled, the model identification procedure is identical to the case without delays.

C. Building a Linear Model From the Koopman Operator

For dynamical systems with inputs, we are interested in using the Koopman operator to construct discrete linear models of the following form:

$$\begin{aligned} z[j+1] &= Az[j] + Bu[j] \\ x[j] &= Cz[j] \end{aligned} \tag{19}$$

for each $j \in \mathbb{N}$, where $x[0]$ is the initial condition in state space, $z[0] = \psi(x[0])$ is the initial lifted state, $u[j] \in \mathbb{R}^m$ is the input at the jth step, and C acts as a projection operator from the lifted space onto the state space. Specifically, we desire a representation in which (nonlifted) inputs appear *linearly*, because models of this form are amenable to real-time convex optimization techniques for feedback control design, as we describe in Section III.

We construct a model of this form by applying the system identification method of Section II-B to the following lifted snapshot pairs with the input appended:

$$\alpha[k] = \begin{bmatrix} \psi(a[k]) \\ u[k] \end{bmatrix} \qquad \beta[k] = \begin{bmatrix} \psi(b[k]) \\ u[k] \end{bmatrix} \tag{20}$$

for each $k \in \{1, \dots, K\}$. The input $u[k]$ in snapshot k is not lifted to ensure that it appears linearly in the resulting model. With these pairs, we define the following $K \times (N+m)$ matrices:

$$\Gamma_\alpha = \begin{bmatrix} \alpha[1]^\top \\ \vdots \\ \alpha[K]^\top \end{bmatrix} \qquad \Gamma_\beta = \begin{bmatrix} \beta[1]^\top \\ \vdots \\ \beta[K]^\top \end{bmatrix} \tag{21}$$

and solve for the corresponding Koopman operator according to (12)

$$\bar{U}_{T_s} := \Gamma_\alpha^\dagger \Gamma_\beta. \tag{22}$$

Note that by (11) and (22), the transpose of this Koopman matrix is the best approximation of a transition matrix between the elements of snapshot pairs in the L^2-norm sense, i.e., $\bar{U}_{T_s} \in \mathbb{R}^{(N+m)\times(N+m)}$ is the minimizer to

$$\min_{U'} \sum_{k=1}^{K} \left\| {U'}^\top \alpha[k] - \beta[k] \right\|_2^2 \tag{23}$$

and we desire the $A \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times m}$ matrices that minimize

$$\min_{A',B'} \sum_{k=1}^{K} \|A'\psi(a[k]) + B'u[k] - \psi(b[k])\|_2^2. \tag{24}$$

Therefore, the best A and B matrices of (19) are embedded in $\bar{U}_{T_s}^\top$ and can be isolated by partitioning it as follows:

$$\bar{U}_{T_s}^\top = \begin{bmatrix} A_{N\times N} & B_{N\times m} \\ O_{m\times N} & I_{m\times m} \end{bmatrix} \tag{25}$$

where I denotes an identity matrix, O denotes a zero matrix, and the subscripts denote the dimensions of each matrix. The C matrix is defined as

$$C = \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \tag{26}$$

since, by (4), $x = [\psi_1(x), \dots, \psi_n(x)]^\top$. Note that we can also incorporate input delays into the model by appending them to the snapshot pairs as we did in (17) and (18). Algorithm 1 summarizes the proposed linear model construction process.

D. Building a Nonlinear Model From the Koopman Operator

The Koopman operator can also be used to identify a continuous nonlinear dynamical model of the form

$$\dot{x}(t) = F(x(t), u(t)). \tag{27}$$

This representation admits nonlinear input terms, so we augment the snapshot pairs to have the input appended

$$a[k] = \begin{bmatrix} x[k] \\ u[k] \end{bmatrix} \tag{28}$$

$$b[k] = \begin{bmatrix} \phi_{T_s}(x[k]) + \sigma[k] \\ u[k] \end{bmatrix} \tag{29}$$

and modify the lifting function to take this augmented state as its input argument $\psi : \mathbb{R}^{n+m} \to \mathbb{R}^N$. We then approximate the Koopman operator just as in Section II-B via (15) and (16).

To construct a model in the form of (27), we introduce the infinitesimal generator of the set of Koopman operators $G : \mathcal{F} \to \mathcal{F}$ [31, eq. (7.6.5)], which is defined in terms of the vector field F as

$$G = F \cdot \nabla_x. \tag{30}$$

This generator describes the dynamics of the observables along trajectories of the system, whereas the set of Koopman operators describes the flow of observables. Framed in terms of the more familiar context of finite-dimensional linear systems, an observable is analogous to the state, G is analogous to the A matrix describing the dynamics of the state, and U_t is analogous to the state-transition matrix for time t. The state-transition matrix of a finite-dimensional linear system is related to its A matrix via the matrix exponential, and similarly, the relationship between the Koopman operator and its generator is given by the following matrix exponential expression:

$$U_t = e^{Gt}. \tag{31}$$

In practice, our goal is to find a function $\bar{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that describes the behavior of the function F as accurately as possible in the L^2-norm sense on the finite-dimensional subspace $\bar{\mathcal{F}}$. With the approximation of the Koopman operator $\bar{U}_{T_s}$ in hand, we can solve for the infinitesimal generator $\bar{G}$ by inverting (31):

$$\bar{G} = \frac{1}{T_s} \log \bar{U}_{T_s} \in \mathbb{R}^{N \times N} \tag{32}$$

where $\log$ denotes the principal matrix logarithm [34, Ch. 11]. Note that the principal matrix logarithm is only defined for matrices whose eigenvalues all have nonnegative real parts, and that $\bar{U}_{T_s}$ may have zero or negative eigenvalues when the number of data points is too small [21, Sec. III-B]. Therefore, this method might fail if the number of data points is insufficient. In this instance, more system measurements can be taken to resolve the issue.

With $\bar{G}$ known, (30) can be used to identify $\bar{F}$. Consider G applied to an observable $f \in \mathcal{F}$. According to (30), this is equivalent to the inner product of the vector field F and the gradient of f with respect to x:

$$Gf(x,u) = \frac{\partial f(x,u)}{\partial x} F(x,u). \tag{33}$$

Again using θ to denote the vector representation of an observable $\bar{f}$ and $a = [x^\top, u^\top]^\top$ to represent the augmented state with the input appended, the finite-dimensional equivalent of (33) is given by

$$(\bar{G}\theta)^T \psi(a) = \theta^T \frac{\partial \psi(a)}{\partial x} \bar{F}(x,u). \tag{34}$$

We seek the vector field $\bar{F}$ such that (34) holds as well as possible in the L^2-norm sense for all observed data. Therefore, we choose the least-squares solution to (34) over the set of all observed data points $\{a_k = [x_k^\top, u_k^\top]^\top | k = 1, ..., K\}$, which is given by

$$\bar{F}(x,u) = \begin{bmatrix} \frac{\partial \psi(a_1)}{\partial x} \\ \vdots \\ \frac{\partial \psi(a_K)}{\partial x} \end{bmatrix}^\dagger \begin{bmatrix} \bar{G}^T \\ \vdots \\ \bar{G}^T \end{bmatrix} \psi(a). \tag{35}$$

Algorithm 2 summarizes this nonlinear system identification process. For a more thorough treatment, see [21] and [22].

E. Practical Considerations: Overfitting and Sparsity

A pitfall of data-driven modeling approaches is the tendency to overfit. While least-squares regression yields a solution that minimizes the total L^2-norm error with respect to the training data, this solution can be particularly susceptible to outliers and noise [35]. To guard against overfitting to noise while identifying $\bar{U}_{T_s}$, we utilize the L^1-regularization method of Least Absolute Shrinkage and Selection Operator (LASSO) [36]

$$\vec{\bar{U}}_{T_s} = \underset{\vec{\bar{U}}_{T_s}}{\arg\min} \quad ||\vec{\Gamma}_\alpha \vec{U}_{T_s} - \vec{\Gamma}_\beta||_2^2 + \lambda ||\vec{U}_{T_s}||_1 \tag{36}$$

where $\lambda \in \mathbb{R}^+$ is the weight of the L^1 penalty term, and $\vec{\cdot}$ denotes a vectorized version of each matrix with dimensions consistent with the stated problem. For $\lambda = 0$, (36) provides the same unique least-squares solution as (22); as λ increases, it drives the elements of $\vec{U}_{T_s}$ to zero. For an overview of the LASSO method and its implementation, see [36].

The benefit of using L^1-regularization to reduce overfitting rather than L^2-regularization (e.g., ridge regression) is its ability to drive elements to zero, rather than just making them small. This promotes sparsity in the resulting Koopman operator matrix (and consequently the A and B matrices of the linear model). Sparsity is desirable since it reduces the memory needed to store these matrices on a computer, enabling a higher dimensional set of basis functions to be used to construct the lifting function ψ.

Though sparsity is desirable, it can carry a cost in prediction accuracy for the lifted linear model described by (19). As illustrated in Fig. 2, the lifting function ψ maps from $\mathbb{R}^n$ to $\mathcal{M}$, but at some time step j, $A\psi(a[j]) + Bu[j]$ may not map onto $\mathcal{M}$. When this happens and we try to simulate our linear model from an initial condition, it may leave the space of legitimate "lifted states" and fail to predict behavior accurately. We, therefore, desire the sparsest model that minimizes the distance from $\mathcal{M}$ at each iteration.

This can be accomplished by applying a projection operator at each time step. For each snapshot pair, the ideal projection operator P should satisfy the following for all k:

$$P\left(A\psi(a[k]) + Bu[k]\right) = \psi(b[k]). \tag{37}$$

To build an approximation to this operator, we construct the following $K \times N$ matrix:

$$\Omega_a := \begin{bmatrix} \left(A\psi(a[1]) + Bu[1]\right)^\top \\ \vdots \\ \left(A\psi(a[K]) + Bu[K]\right)^\top \end{bmatrix}. \tag{38}$$

Then, the best projection operator in the L^2-norm sense based on our data is given by

$$P := \left(\Omega_a^\dagger \Psi_b\right)^\top. \tag{39}$$

Composing P with the A and B matrices in (19) yields a modified linear model that significantly reduces the distance

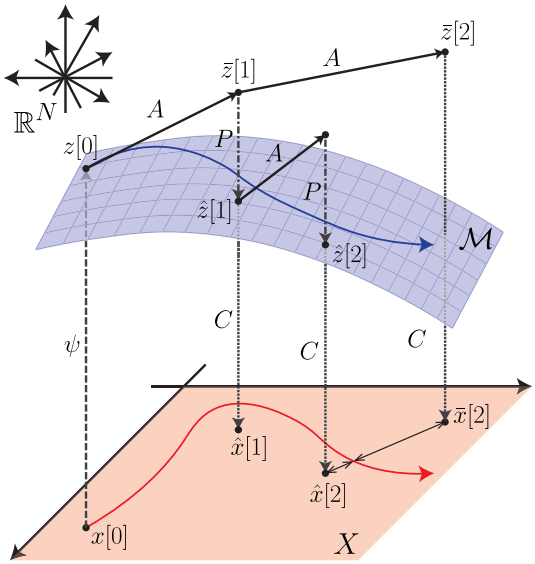


Fig. 2. Illustration of the effect of deviating from the image of the lifted functions $\mathcal{M}$ and how it can be remedied by defining a projection operation, as described in Section II-E. The evolution of the finite-dimensional system in the state space X from $x[0]$ is depicted as a red curve. The lifted version of this evolution is depicted as the blue curve, which is contained in $\mathcal{M}$. The discrete-time system representation in the higher dimensional space created by iteratively applying the state matrix A to $z[j]$ may generate a solution that is outside of $\mathcal{M}$. Though one can still apply C to $\bar{z}$ to project it back to X, this may result in poor performance. Instead, by projecting $\bar{z}[j]$ onto the manifold at each discrete time step to define a new lifted state $\hat{z}[j]$, the deviation from $\mathcal{M}$ is reduced, which improves overall predictive performance.

Algorithm 1: Koopman Linear System Id.

Input: λ , $\{a[k], b[k]\}$ and $u[k]$ for $k = 1, ..., K$
Step 1: Lift data via (6)
Step 2: Combine lifted data and inputs via (20)
Step 3: Approx. Koopman operator $\bar{U}_{T_s}$ via (22) or (36)
Step 4: Extract model matrices A, B via (25)
Step 5: (optional) Identify projection operator P via (39)
Output: A, B, $C := \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix}$,
(optional) $\hat{A} := PA$, $\hat{B} := PB$

from $\mathcal{M}$ at each iteration

$$z[j+1] = \hat{A}z[j] + \hat{B}u[j] \tag{40}$$

where $\hat{A} := PA$ and $\hat{B} := PB$. An unfortunate side effect of applying the projection operator P is an increase in matrix density. However, this does not completely negate the sparsity benefits of L^1-regularization. As we show in Fig. 6 in Section IV, for a real soft robotic system, the resulting $\hat{A}$ matrix is still sparser than the one identified using pure least-squares regression, but is nearly identical in accuracy. Algorithm 1 summarizes the proposed linear model construction process with this added projection.

III. Model Predictive Control

A system model enables the design of model-based controllers that leverage model predictions to choose suitable control inputs

Algorithm 2: Koopman Nonlinear System Id. [21].

Input: λ , $\{a[k], b[k]\}$ and $u[k]$ for $k = 1, ..., K$
Step 1: Lift data via (6)
Step 2: Combine lifted data and inputs via (20)
Step 3: Approx. Koopman operator $\bar{U}_{T_s}$ via (22) or (36)
Step 4: Identify infinitesimal generator $\bar{G}$ by (34)
Step 5: Solve for vector field $\bar{F}$ via (35)
Output: $\bar{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$

for a given task. In particular, model-based controllers can anticipate future events, allowing them to optimally choose control inputs over a finite time horizon. A popular model-based control design technique is model predictive control (MPC), wherein one optimizes the control input over a finite time horizon, applies that input for a single time step, and then optimizes again, repeatedly [37].

This section introduces several MPC controllers based on models constructed from the Koopman operator. Section III-A describes the MPC optimization problem for a linear Koopman model in the form of (19). Section III-B describes the MPC optimization problem for a nonlinear Koopman model in the form of (27). Section III-C discusses the relative computational efficiency of each approach.

A. Linear MPC

For linear systems, MPC consists of iteratively solving a convex quadratic program. Importantly, this is also the case for Koopman-based linear MPC, which we refer to by the abbreviation K-MPC, wherein one solves for the optimal sequence of control inputs over a receding horizon according to the following quadratic program at each time instance k of the closed-loop operation:

$$\begin{aligned}
&\min_{\substack{\{u[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{z[i]\in\mathbb{R}^N\}_{i=0}^{N_h}}} z[N_h]^T G[N_h] z[N_h] + g[N_h]^T z[N_h] + \\
&+ \sum_{i=0}^{N_h-1} (z[i]^T G[i] z[i] + u[i]^T H[i] u[i] + \\
&+ g[i]^T z[i] + h[i]^T u[i]) \\
&\text{s.t. } z[i+1] = \hat{A}z[i] + \hat{B}u[i] \quad \forall \{i\}_{i=0}^{N_h-1} \\
&E[i]z[i] + F[i]u[i] \le b[i] \quad \forall \{i\}_{i=0}^{N_h-1} \\
&z[0] = \psi(x[k])
\end{aligned} \tag{41}$$

where $N_h \in \mathbb{N}$ is the prediction horizon, $G[i] \in \mathbb{R}^{N\times N}$ and $H[i] \in \mathbb{R}^{m\times m}$ are positive-semidefinite matrices, and where each time the program is called, the predictions are initialized from the current lifted state $\psi(x[k])$. The matrices $E[i] \in \mathbb{R}^{c\times N}$ and $F[i] \in \mathbb{R}^{c\times m}$ and the vector $b[i] \in \mathbb{R}^c$ define state and input polyhedral constraints, where c denotes the number of imposed constraints. While the size of the cost and constraint matrices in (41) depends on the dimension of the lifted state N, Korda and Mezić [26] show that these can be rendered independent

Algorithm 3: Koopman-Based Linear MPC [26].

Input: Prediction horizon: N_h
Cost matrices: G_i, H_i, g_i, h_i for $i = 0, ..., N_h$
Constraint matrices: E_i, F_i, b_i for $i = 0, ..., N_h$
Model matrices: $\hat{A}, \hat{B}$
for $k = 0, 1, 2, ...$ **do**
Step 1: Set $z[0] = \psi(x[k])$
Step 2: Solve (41) to find optimal input $(u[i]^*)_{i=0}^{N_h}$
Step 3: Set $u[k] = u[0]^*$
Step 4: Apply $u[k]$ to the system

of N by transforming the problem into its so-called dense form. Algorithm 3 summarizes the closed-loop operation of this Koopman-based linear MPC.

B. Nonlinear MPC

To control a soft robot using NMPC, a reference trajectory is assumed to be given, $x_{\text{ref}} : [t_0, t_f] \to \mathbb{R}^n$, where $0 \leq t_0 < t_f$, the initial position of the robot $x_0 \in \mathbb{R}^n$ is assumed known, and we solve the following optimal control problem:

$$\begin{aligned} \min_{u \in L^2([t_0,t_f];\mathbb{R}^m)} \quad & \int_{t_0}^{t_f} \Big((x(t) - x_{\text{ref}}(t))^T Q (x(t) - x_{\text{ref}}(t)) \Big) + \\ & + u(t)^T R u(t)) \, dt \\ \text{s.t.} \quad & \dot{x}(t) = F(x(t), u(t)) \quad \forall t \in [t_0, t_f] \\ & x(t_0) = x_0 \\ & L(x(t), u(t)) \leq 0_q \quad \forall t \in [t_0, t_f] \end{aligned} \tag{42}$$

where $L^2([t_0, t_f]; \mathbb{R}^m)$ is the space of square integrable functions from $[t_0, t_f]$ to $\mathbb{R}^m$, $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ are positive-semidefinite matrices, $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$ defines state and input constraints, and 0_q is the vector of all 0s with q-elements. We solve this problem by applying a commercial optimal control problem solver—GPOPS-II [38].

Solving (42) quickly enough for real-time receding horizon control is often not possible. In practice, the optimal control inputs must be computed offline and then applied to the system in open loop. Without feedback, the controller is incapable of overcoming model error that causes performance to degrade as the time horizon increases. To counteract this shortcoming, an NMPC controller can be supplemented with a linear feedback controller based on the local linearization of the nonlinear model about the reference trajectory (see [39, Sec. 5.4]). At each time step k, the optimal input $u^*[k]$ is then given by the sum of the open-loop control input computed by solving the aforementioned NMPC problem offline, denoted $\overline{u}[k]$, and a feedback term based on the local linearization that is computed online, denoted $\delta_u[k]$,

$$u^*[k] = \overline{u}[k] + \delta_u[k] \tag{43}$$

where $\delta_u[k]$ is chosen such that it drives the locally linearized dynamics to zero. In practice, we compute $\delta_u[k]$ at each time step by solving a linear MPC program with structure similar to (41).

C. MPC Versus NMPC

Convex optimization programs, i.e., those which have a convex cost function and convex feasible set, have global extrema and can be solved reliably and efficiently [40]. Nonconvex optimization programs, on the other hand, may exhibit multiple local extrema, and there is no computationally tractable technique for finding global solutions to generic problems. Typically, nonconvex optimization methods only find local solutions that depend on an initial guess. Thus, for real-time optimal control problems, convexity is desirable. A major advantage of lifting and identifying a linear system model using the Koopman operator is that the MPC problem for such models is convex.

The MPC optimization problem described in Section III-A is convex because it has a quadratic cost function and model dynamics that are described by linear constraints. Since it is convex, it has a unique globally optimal solution that can efficiently be identified without initialization for models with thousands of states and inputs [40]–[42]. The NMPC optimization problem described in Section III-B also has a quadratic cost function, but it is not convex because it has model dynamics that are described by nonlinear constraints. As a result, algorithms to solve this problem typically require initialization and can struggle to find globally optimal solutions [43]. Though techniques have been proposed to improve the speed of such algorithms [38], [44] or even globally solve NMPC problems without requiring initialization [45], these formulations still take several seconds per iteration, which make them too slow to be applied in most real-time control scenarios. This difference is highlighted in Section V via computation time comparisons between MPC and NMPC for a soft robot.

IV. Application: Control of a Soft Robot Arm

This section describes the soft robot platform and the set of experiments used to demonstrate the efficacy of the modeling and control methods described in Sections II and III. Footage from the final experiment of the soft robot performing several tasks can be found in a supplementary video file,[1] and code used to construct Koopman-based models and controllers from data can be found in a publicly accessible repository.[2]

A. Robot Description: Soft Arm With a Laser Pointer

We experimentally evaluated the Koopman control approach on a soft robot arm performing the task of drawing shapes with a laser pointer, similar to the handwriting task described in [46]. The robot is a suspended soft arm with a laser pointer attached to the end-effector (see Fig. 3). The laser dot is projected onto a 50×50 cm flat board, which sits 34 cm beneath the tip of the laser pointer when the robot is in its relaxed position (i.e., hanging straight down). The position of the laser dot is measured by a digital webcam overlooking the board.

The arm consists of two bending sections. The interior of each section is composed of three pneumatic artificial muscles

[1] [Online]. Available: https://youtu.be/1-XSDGHKous
[2] [Online]. Available: https://github.com/ramvasudevan/soft-robot-koopman

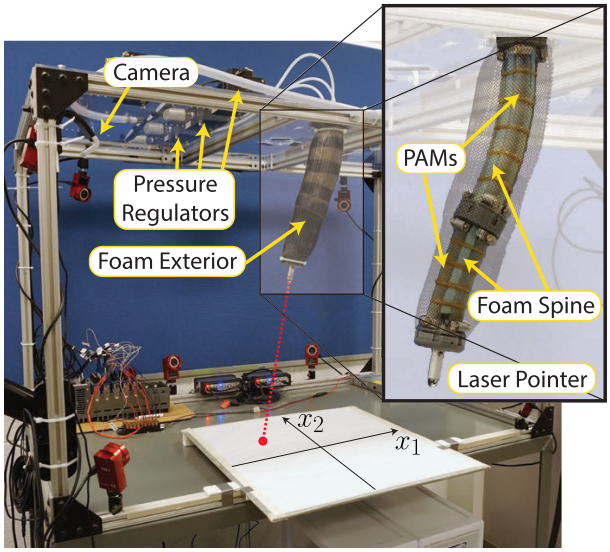


Fig. 3. Soft robot consists of two bending segments encased in a foam exterior with a laser pointer attached to the end-effector. A set of three pressure regulators is used to control the pressure inside of the PAMs, and a camera is used to track the position of the laser dot.

or pneumatic actuators (PAMs) (also known as McKibben actuators [47]) adhered to a central foam spine by latex rubber bands (see Fig. 3). The exterior is composed of polyurethane foam, which serves to dampen high-frequency oscillations. The PAMs in the upper and lower sections are internally connected so that only three input pressure lines are required, and they are arranged such that for any bending of the upper section, bending in the opposite direction occurs in the lower section. This ensures that the laser pointer mounted to the end-effector points approximately vertically downward and the laser light strikes the board at all times. The pressures inside the actuators are regulated by three Enfield TR-010-g10-s pneumatic pressure regulators, which accept 0–10-V command signals corresponding to pressures of approximately 0–140 kPa. In the experiments, the input is three-dimensional and corresponds to the voltages applied to the three pressure regulators. The state is two-dimensional and corresponds to the position of the laser dot with respect to the center of the board in Cartesian coordinates.

B. Characterization of Dynamic Uncertainty

Most real mechanical systems exhibit some dynamic uncertainty (i.e., an identical input and state may produce a slightly different output). Electronic pressure regulators demonstrate this type of behavior, which can limit the precision of pneumatically driven soft robotic systems and undermine the predictive capability of models.

We quantified the dynamic uncertainty of our soft robot system by observing the variations in output from period-to-period under sinusoidal inputs to the three actuators of the form

$$u[k] = \begin{bmatrix} 6\sin(\frac{2\pi}{T}kT_s) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{T}{3}) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{2T}{3}) + 3 \end{bmatrix} \quad (44)$$

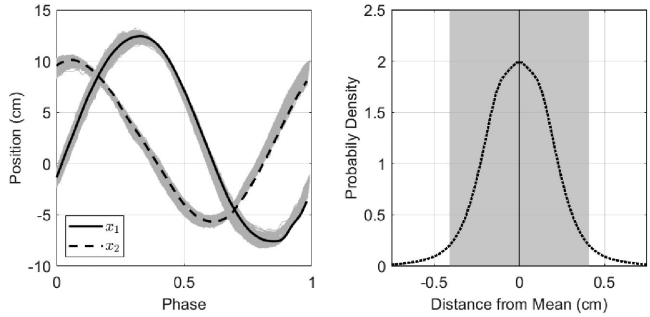


Fig. 4. Left plot shows the average response of the soft robot system over a single period when the sinusoidal inputs of varying frequencies described by (44) are applied. All of the particular responses are subimposed in light gray. The right plot shows the distribution of trajectories about the mean, with all distances within two standard deviations (0.43 cm) highlighted in gray. The width of the distribution illustrates how it is possible for identical inputs to produce outputs that vary by almost 1 cm.

for periods of $T = 6, 7, 8, 9, 10, 11, 12$ s and a sampling time of $T_s = 0.083$ s with a zero-order hold between samples. Under these inputs, the laser dot traces out a circle with some variability in the trajectory over each period. In Fig. 4, the trajectories over 210 periods are superimposed along with the average over all trials. The standard deviation of this distribution is 0.215 cm. This inherent uncertainty will limit the tracking performance of the system, independent of the employed controller.

C. Data Collection and Model Identification

Data for constructing models were collected over 12 trials lasting approximately 5 min each. A randomized "ramp and hold" type input was applied during each trial to generate a representative sampling of the system's behavior over its entire operating range. To ensure randomization, a matrix $\Upsilon \in [0, 10]^{3\times 1000}$ of uniformly distributed random numbers between 0 and 10 was generated to be used as an input lookup table. Each control input was varied between elements in consecutive columns of the table over a transition period T_u, with a time offset of $T_u/3$ between each of the three control signals, and with a sampling time of $T_s = 0.083$ s with a zero-order hold between samples

$$u_i[k] = \frac{(\Upsilon_{i,j+1} - \Upsilon_{i,j})}{T_u}\left(kT_s + \frac{(i-1)T_u}{3}\right) + \Upsilon_{i,j} \quad (45)$$

where $j = \text{floor}(kT_s/T_u)$ is the current index into the lookup table at time t. Three trials were conducted using each of the transition periods $T_u = 2, 3, 4, 5$ s.

Four models were fit from data: a linear state-space model using the subspace method [48], a linear Koopman model using the approach from Section II-C, a nonlinear Koopman model using the approach from Section II-D, and a nonlinear autoregressive with external input (NARX) neural network model identified using the Levenberg–Marquardt backpropogation algorithm. Each of these models was trained from the randomly generated data just once, independent of any specific task.

The linear state-space model provides a baseline for comparison and was identified from the same data as the Koopman models using the MATLAB System Identification Toolbox [49]. This model is a four-dimensional linear state-space model expressed in observer canonical form.

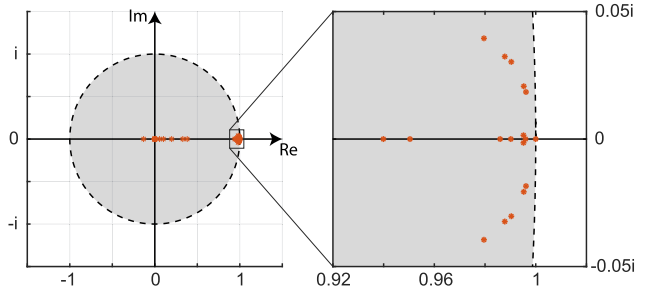


Fig. 5. Eigenvalues of the $\hat{A}$ matrix of the linear Koopman model all lie within or on the boundary of the complex unit disk, indicating that the discrete linear Koopman model is marginally stable.

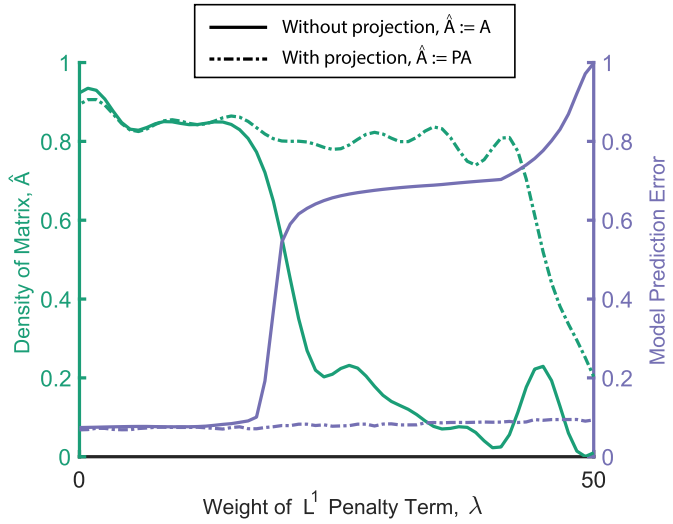


Fig. 6. As the weight of λ [the L^1 penalty term in (36)] increases, the density of the lifted system matrix $\hat{A}$ decreases compared to the least-squares solution, which occurs at $\lambda = 0$. If the projection operator P [defined in (39)] is not applied (shown by the solid lines), this decrease in density produces a large increase in model prediction error. If the projection operator P is applied (shown by the dashed lines), the decrease in density is less significant, but the increase in model prediction error is negligible. The model prediction error refers to the average Euclidean distance between the predicted and measured trajectory normalized such that a constant prediction of zeros corresponds to an error of 1.

The linear Koopman model was identified on a set of 45,103 snapshot pairs $\{a[k], b[k]\}$ that incorporate a single delay $d = 1$:

$$a[k] = \begin{bmatrix} x[k]^\top & x[k-1]^\top & u[k-1]^\top \end{bmatrix}^\top \tag{46}$$

$$b[k] = \begin{bmatrix} (\phi_{T_s}(x[k]) + \sigma[k])^\top & x[k]^\top & u[k]^\top \end{bmatrix}^\top \tag{47}$$

and used an $N = 36$ dimensional set of basis functions consisting of all monomials of maximum degree 2. To find the sparsest acceptable matrix representation of the Koopman operator, (36) was solved for $\lambda = 0, 1, 2, ..., 50$. Predictions from the resulting models were evaluated against a subset of the training data, with the error quantified as the average Euclidean distance between the prediction and actual trajectory at each point, normalized by the average Euclidean distance between the actual trajectory and the origin. Fig. 6 shows that as λ increases so does this error, but the density of the $\hat{A}$ matrix of the lifted linear model decreases. The model chosen used a value of $\lambda = 50$, because it yielded an $\hat{A}$ matrix with roughly 80% of its elements equal to zero without significantly increasing the model prediction error.

TABLE I
AVERAGE PREDICTION ERROR UNDER SINUSOIDAL INPUTS (IN CENTIMETERS)

	Model	Period of Sinusoidal Inputs (seconds)							Avg.
		6	7	8	9	10	11	12	
Linear	Koop. (linear)	2.12	2.08	2.00	1.98	1.89	1.82	1.74	1.92
	State Space	5.56	5.17	4.97	4.83	4.64	4.44	4.24	4.74
Nonlinear	Koop. (nonlinear)	1.35	1.47	1.56	1.63	1.70	1.69	1.69	1.61
	Neural Network	3.23	3.29	3.36	3.31	3.20	3.07	2.94	3.18

The eigenvalues of $\hat{A}$ are plotted on the complex plane in Fig. 5. All of the eigenvalues lie inside or on the unit circle, indicating that the discrete dynamical system described by $\hat{A}$ is marginally stable [39]. This is consistent with our intuition regarding the behavior of the soft robot arm. In the absence of a control input, the state returns to a neighborhood of the origin from any initial condition, but does not necessarily converge to the origin itself. This lack of asymptotic stability can be attributed to robot's foam exterior, which introduces hysteresis.

The nonlinear Koopman model was identified on a set of 45,103 snapshot pairs $\{a[k], b[k]\}$ that have the input appended, but do not incorporate any delays:

$$a[k] = \begin{bmatrix} x[k]^\top & u[k]^\top \end{bmatrix}^\top \tag{48}$$

$$b[k] = \begin{bmatrix} (\phi_{T_s}(x[k]) + \sigma[k])^\top & u[k]^\top \end{bmatrix}^\top \tag{49}$$

and used an $N = 35$ dimensional set of basis functions consisting of all monomials of maximum degree 3. To construct the Koopman operator matrix for this model, (36) was solved with $\lambda = 0$, which corresponds to the least-squares solution.

The NARX neural network model was identified using the same set of 45,103 snapshot pairs as the linear Koopman model, which incorporates a single delay $d = 1$. The model was trained using the MATLAB Neural Network Toolbox [49] with sigmoid activation functions, and the number of hidden neurons was tuned from 5 to 20. The best results were achieved with ten hidden neurons, so this is what was used for the model prediction comparison described in Section IV-D.

D. Experiment 1: Model Prediction Comparison

The accuracy of the predictions generated by each of the four models was evaluated by comparing them to the actual behavior of the system under the sinusoidal inputs defined in (44). These comparison data were not part of the training set. The model responses were simulated over one period of the sinusoidal inputs given the same initial condition and input as the real system. Table I displays the average Euclidean distance between the predicted laser dot position and the actual position at each point in time, and Fig. 7 shows the tracking performance for the case when the inputs have period $T = 6$ s. These results

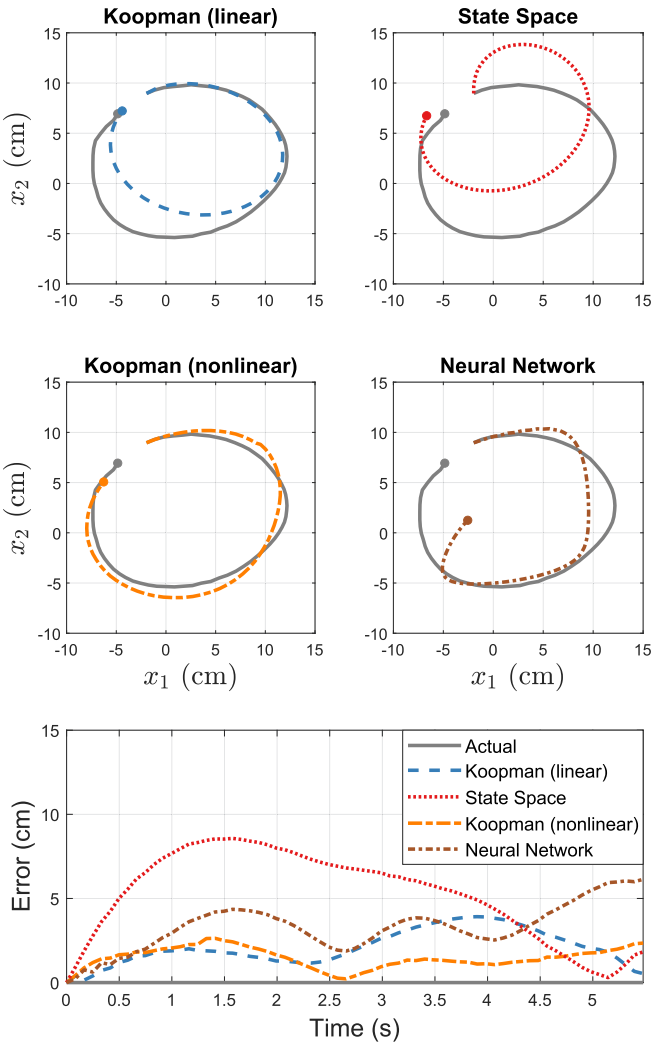


Fig. 7. Actual response and the model predictions for the robot with the sinusoidal inputs described in (44) with period $T = 6$ seconds applied. The left plot shows the actual trajectory of the laser dot along with model predictions. The error displayed on the bottom plot is defined as the Euclidean distance between the predicted laser dot position and the actual position at each point in time.

illustrate that both Koopman models generate more accurate predictions than the state-space and neural network models with the nonlinear Koopman model producing slightly better predictions than the linear Koopman model.

E. Experiment 2: Model-Based Control Comparison

Three of the identified models were used to construct four model predictive controllers denoted by the following abbreviations:

- L-MPC: linear MPC based on the linear state-space model;
- K-MPC: linear MPC based on the linear Koopman model;
- K-NMPC: NMPC based on the nonlinear Koopman model;
- K-NMPC+LL: NMPC based on the nonlinear Koopman model plus a linear feedback term.

The neural network model was not used to construct an NMPC controller. Due to the inherent computational advantages of linear MPC, we were only interested in comparing it against the best-case version of NMPC. The nonlinear Koopman model demonstrated superior prediction accuracy in Experiment 1; therefore, we considered the inclusion of a neural-network-based NMPC to be superfluous.

The two controllers based on linear models (L-MPCand K-MPC) both solve an optimization problem in the form of (41) at each time step using the Gurobi Optimization software [50]. They run in closed loop at 12 Hz, feature an MPC horizon of 2 s ($N_h = 24$), and a cost function that penalizes deviations from a reference trajectory $y^{\text{ref}}[k]$ over the horizon with both a running and terminal cost:

$$\begin{aligned}\text{Cost} = 100\left(y[N_h] - y^{\text{ref}}[N_h]\right)^\top \left(y[N_h] - y^{\text{ref}}[N_h]\right) + \\ + \sum_{i=0}^{N_h-1} 0.1\left(y[i] - y^{\text{ref}}[i]\right)^\top \left(y[i] - y^{\text{ref}}[i]\right).\end{aligned} \tag{50}$$

In the L-MPC case, $y[i] = C_L x_L[i]$, where x_L is the four-dimensional system state and C_L is the projection matrix that isolates the states describing the current laser dot coordinates. In the K-MPC case, $y[i] = Cz[i]$, where C is defined as in (26).

The two controllers based on the nonlinear Koopman model (K-NMPC and K-NMPC+LL) compute open-loop inputs offline that refresh at a rate of 2 Hz for an entire task by solving an optimization problem in the form of (42) offline, with $Q = 100 \times I_{2\times 2}, R = I_{3\times 3}$. The K-NMPC then applies these inputs to the system in open loop without any feedback. The K-NMPC+LL controller utilizes the same open-loop inputs, but also computes online feedback based on a local linearization of the nonlinear Koopman model about the reference trajectory and applies inputs in the form of (43) at a rate of 10 Hz. In all four controllers, the inputs are constrained to be in [0,10], since a voltage outside of this interval is not a valid command signal into the pressure regulators.

The tracking performance of the controllers was assessed with respect to a set of three trajectory following tasks. Each task was to follow a reference trajectory as it traced out one of the following shapes over a specified amount of time:

1) Task 1: Pac-Man (90 s);
2) Task 2: Star (120 s);
3) Task 3: Block letter M (180 s).

The error for each trial was quantified as the average Euclidean distance from the reference trajectory at each time step over the length of the trial

$$\text{Error} = \frac{\sum_{i=1}^{N\text{steps}} \sqrt{\left(y[i] - y^{\text{ref}}[i]\right)^\top \left(y[i] - y^{\text{ref}}[i]\right)}}{N_{\text{steps}}} \tag{51}$$

where N_{steps} denotes the total number of time steps in a trial. The performances of all four controllers in completing tasks 1–3 are shown visually in Fig. 8, and the error for each trial is shown in Table II.

The K-NMPC open-loop control inputs took 2.80, 11.57, and 15.82 h to compute offline for tasks 1–3, respectively, using GPOPS-II on a high-end computer setup with 1-TB RAM and

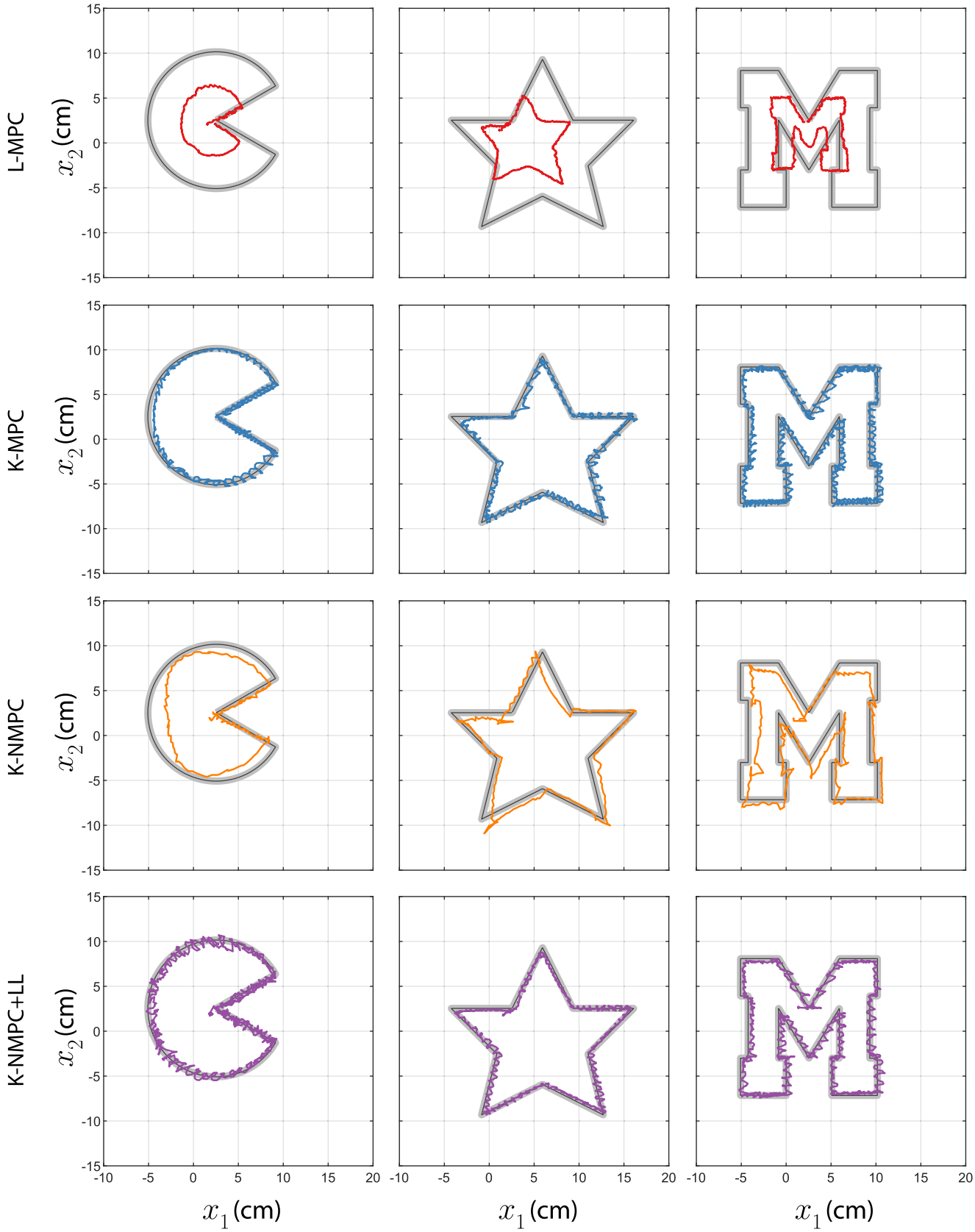


Fig. 8. Results of the L-MPC (row 1, red), the K-MPC (row 2, blue), the K-NMPC (row 3, orange), and the K-NMPC+LL controller (row 4, purple) in performing trajectory-following tasks 1–3. The reference trajectory for each task is subimposed in black as well as a gray buffer with width equal to two standard deviations of the noise probability density shown in Fig. 4.

TABLE II
AVERAGE EUCLIDEAN DISTANCE ERROR IN TRAJECTORY-FOLLOWING TASKS (IN CENTIMETERS)

Controller	Task			Avg.	Std. Dev.
	1	2	3		
L-MPC	3.25	3.86	3.25	3.45	0.35
K-MPC	0.44	0.52	0.43	0.46	0.05
K-NMPC	0.95	0.96	0.99	0.96	0.02
K-NMPC-LL	0.43	0.37	0.39	0.40	0.03

144 CPUs running at 2.4 GHz. The L-MPC and K-MPC optimization problems were solved online repeatedly in less than 0.083 s for a 2-s receding horizon on a computer with 16-GB RAM and a 3.6-GHz CPU, requiring no offline computations.

V. DISCUSSION

In all three tasks, the K-NMPC+LL controller achieved the best tracking performance, exhibiting an overall average error of 0.40 cm, followed closely by the K-MPC controller with an average error of 0.46 cm. The K-NMPC controller exhibited a larger average error of 0.96 cm as it was unable to correct for errors online without the assistance of feedback. The L-MPC controller exhibited the worst tracking performance with an average error of 3.45 cm, which is more than three times larger than the average error of any of the Koopman-based controllers.

The K-NMPC controller had the lowest standard deviation in its error, which is evident by the relative smoothness of its trajectories compared to those of the other controllers. This can likely be attributed to the fact that the K-NMPC controller optimized the control input over the entire trial at once and did not have any way to correct for deviations online. The other controllers use feedback to try to correct for tracking errors online, which results in some overshooting and oscillations about the desired trajectory. In some applications, the reduced accuracy of the K-NMPC controller may be preferable to the higher frequency behavior of the other controllers. It should be noted, however, that this higher frequency behavior could likely be reduced by tuning the MPC cost function parameters to eliminate overshoot. Therefore, the oscillatory behavior observed in these experiments does not necessarily reflect a fundamental feature of the proposed MPC algorithms themselves, but is likely just an artifact of this particular choice of cost function.

Considering that the standard deviation under repeated inputs as described in Section IV-B is 0.215 cm, even a perfect controller would be expected to have an average error of approximately 0.215 cm. If we normalize that by the 50-cm width of the robot's square-shaped workspace, it amounts to an error of $4.3 \times 10^{-3}\%$. Normalized the same way, the average errors exhibited by the K-MPC and K-NMPC+LL controllers are $9.2 \times 10^{-3}\%$ and $8.0 \times 10^{-3}\%$, respectively. Hence, their performance may be considered on par with what is expected from a perfect controller in the sense that their normalized error is of the same order of magnitude. In contrast, the normalized L-MPC error is $6.9 \times 10^{-2}\%$, a full order of magnitude larger than that of the other controllers.

The poor performance of the L-MPC controller can be attributed to the inaccuracy of the linear state-space model upon which it was based, since it is identical to the K-MPC controller in every other way. This should not be surprising considering the results of Experiment 1, in which the linear state-space model consistently provided the worst predictions over a 2-s horizon (shown in Table I).

While the K-NMPC+LL controller achieved a slightly better tracking performance than the K-MPC controller, the K-MPC controller would still be preferable for most applications due to its vastly superior computational efficiency. The K-NMPC optimization problem took so long to solve (> 2 h) that its solution had to be computed offline, whereas the K-MPC optimization problem could be repeatedly solved in less than 0.083 s over a 2-s receding horizon online. The K-MPC controller is also capable of adapting to a changing reference trajectory online, since it does not rely on linearizations about a predetermined path. This should make it much more reliable in the presence of external disturbances.

VI. CONCLUSION

In this article, a data-driven modeling and control method based on Koopman operator theory was successfully applied to a soft robot. Three Koopman-based MPC controllers were shown to be capable of commanding a soft robot to accurately follow a reference trajectory better than an MPC controller based on another linear data-driven model. By making it possible to construct accurate control-oriented models of soft robots from data when no physics-based model is available, this method enables the rapid development of new control strategies and applications.

This work also demonstrated how the Koopman approach can simplify the control of nonlinear systems even when a good model is available. While the nonlinear model predictive control achieved the highest accuracy in our experiments, the Koopman-based linear controller was able to achieve nearly identical accuracy with significantly lower computational demand. Koopman operator theory offers a linearization method that does not suffer from the decline in predictive performance normally associated with linearization, making accurate linear control of nonlinear systems achievable.

While these preliminary results are promising, further work is needed to make such methods feasible for higher dimensional robotic systems. In this particular case, the number of observed states needed to uniquely describe the configuration of the robot is quite low. More complicated systems may require a higher number of observed states, which, in turn, would yield a much higher dimensional Koopman model. This dimensionality issue is the primary motivation for the sparsity promoting methods described in Section II-E. Additional work will explore strategies for further promoting sparsity, choosing the most effective basis of observables and building models that can account for external loading and contact forces.

REFERENCES

[1] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, no. 7553, pp. 467–475, 2015.
[2] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Soft robotics for chemists," *Angewandte Chemie*, vol. 123, no. 8, pp. 1930–1935, 2011.

[3] M. T. Tolley *et al.*, "A resilient, untethered soft robot," *Soft Robot.*, vol. 1, no. 3, pp. 213–223, 2014.

[4] A. D. Marchese, C. D. Onal, and D. Rus, "Autonomous soft robotic fish capable of escape maneuvers using fluidic elastomer actuators," *Soft Robot.*, vol. 1, no. 1, pp. 75–87, 2014.

[5] M. Spong and M. Vidyasagar, *Robot Dyn. Control*. New Delhi, India: Wiley, 2008.

[6] F. Allgöwer and A. Zheng, *Nonlinear Model Predictive Control*, vol. 26. Cambridge, MA, USA: Birkhäuser, 2012.

[7] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robot. Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[8] A. R. Ansari and T. D. Murphey, "Sequential action control: Closed-form optimal control for nonlinear and nonsmooth systems," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1196–1214, Oct. 2016.

[9] R. J. Webster III and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *Int. J. Robot. Res.*, vol. 29, no. 13, pp. 1661–1683, 2010.

[10] L. L. Howell, A. Midha, and T. Norton, "Evaluation of equivalent spring stiffness for use in a pseudo-rigid-body model of large-deflection compliant mechanisms," *J. Mech. Des.*, vol. 118, no. 1, pp. 126–131, 1996.

[11] D. Bruder, A. Sedal, R. Vasudevan, and C. D. Remy, "Force generation by parallel combinations of fiber-reinforced fluid-driven actuators," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3999–4006, Oct. 2018.

[12] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Trans. Robot.*, vol. 35, no. 1, pp. 124–134, Feb. 2019.

[13] I. A. Gravagne, C. D. Rahn, and I. D. Walker, "Large deflection dynamics and control for planar continuum robots," *IEEE/ASME Trans. Mechatronics*, vol. 8, no. 2, pp. 299–307, Jun. 2003.

[14] D. Trivedi, A. Lotfi, and C. D. Rahn, "Geometrically exact models for soft robotic manipulators," *IEEE Trans. Robot.*, vol. 24, no. 4, pp. 773–780, Aug. 2008.

[15] A. Sedal, A. Wineman, R. B. Gillespie, and C. D. Remy, "Comparison and experimental validation of predictive models for soft, fiber-reinforced actuators," *Int. J. Robot. Res.*, to be published, doi: 10.1177/0278364919879493.

[16] D. Bruder, A. Sedal, J. Bishop-Moser, S. Kota, and R. Vasudevan, "Model based control of fiber reinforced elastofluidic enclosures," in *Proc. IEEE Int. Conf. Robot. Autom.* 2017, pp. 5539–5544.

[17] A. Sedal, D. Bruder, J. Bishop-Moser, R. Vasudevan, and S. Kota, "A constitutive model for torsional loads on fluid-driven soft robots," in *Proc. Int. Des. Eng. Tech. Conf. Comput. Inf. Eng. Conf*, 2017, Art. no. V05AT08A016.

[18] J. Bishop-Moser, G. Krishnan, C. Kim, and S. Kota, "Design of soft robotic actuators using fluid-filled fiber-reinforced elastomeric enclosures in parallel combinations," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst*, 2012, pp. 4264–4269.

[19] M. T. Gillespie, C. M. Best, E. C. Townsend, D. Wingate, and M. D. Killpack, "Learning nonlinear dynamic models of soft robots for model predictive control with neural networks," in *Proc. IEEE Int. Conf. Soft Robot*, 2018, pp. 39–45.

[20] D. Bruder, C. D. Remy, and R. Vasudevan, "Nonlinear system identification of soft robot dynamics using Koopman operator theory," in *Proc. Int. Conf. Robot. Autom*, 2019, pp. 6244–6250.

[21] A. Mauroy and J. Goncalves, "Linear identification of nonlinear systems: A lifting technique based on the Koopman operator," in *Proc. IEEE 55th Conf. Decis. Control (CDC)*, 2016.

[22] A. Mauroy and J. Goncalves, "Koopman-based lifting techniques for nonlinear systems identification," *IEEE Trans. Autom. Control*, 2019, pp. 2550–2565.

[23] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PLoS One*, vol. 11, no. 2, 2016, Art. no. e0150171.

[24] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Generalizing Koopman theory to allow for inputs and control," *SIAM J. Appl. Dyn. Syst.*, vol. 17, no. 1, pp. 909–930, 2018.

[25] I. Abraham, G. de la Torre, and T. Murphey, "Model-based control using Koopman operators," in *Proc. Robot.: Sci. Syst. Conf.*, Cambridge, MA, USA, Jul. 2017.

[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] I. Abraham and T. D. Murphey, "Active learning of dynamics for data-driven control using Koopman operators," *IEEE Trans. Robot.*, vol. 35, no. 5, pp. 1071–1083, Oct. 2019.

[28] G. Mamakoukas, M. Castano, X. Tan, and T. Murphey, "Local Koopman operators for data-driven control of robotic systems," in *Proc. Robot.: Sci. Syst. Conf.*, 2019.

[29] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," *J. Nonlinear Sci.*, vol. 25, no. 6, pp. 1307–1346, 2015.

[30] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, "Modeling and control of soft robots using the Koopman operator and model predictive control," in *Proc. Robot.: Sci. Syst. Conf.*, Freiburg im Breisgau, Germany, Jun. 2019.

[31] A. Lasota and M. C. Mackey, *Chaos, Fractals, and Noise: Stochastic Aspects of Dynamics*, vol. 97. Berlin, Germany: Springer, 2013.

[32] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos: Interdiscipl. J. Nonlinear Sci.*, vol. 22, no. 4, 2012, Art. no. 047510.

[33] R. Penrose, "On best approximate solutions of linear matrix equations," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 52. Cambridge, U.K.: Cambridge Univ. Press, 1956, pp. 17–19.

[34] N. J. Higham, *Functions of Matrices: Theory and Computation*, vol. 104. Philadelphia, PA, USA: SIAM, 2008.

[35] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, vol. 589. Hoboken, NJ, USA: Wiley, 2005.

[36] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *J. Roy. Statist. Soc.*, vol. 58, no. 1, pp. 267–288, 1996.

[37] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Madison, WI, USA: Nob Hill Pub., 2009.

[38] M. A. Patterson and A. V. Rao, "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Trans. Math. Softw.*, vol. 41, no. 1, 2014, Art. no. 1.

[39] K. J. Aström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton Univ., 2010.

[40] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge Univ. Press, 2004.

[41] J. A. Paulson, A. Mesbah, S. Streif, R. Findeisen, and R. D. Braatz, "Fast stochastic model predictive control of high-dimensional systems," in *Proc. IEEE 53rd Annu. Conf. Decis. Control*, 2014, pp. 2802–2809.

[42] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," in *Proc. 12th Int. Conf. Control*, 2018, pp. 339–339.

[43] E. Polak, *Optimization: Algorithms and Consistent Approximations*, vol. 124. Berlin, Germany: Springer, 2012.

[44] A. Hereid and A. D. Ames, "FROST: Fast robot optimization and simulation toolkit," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 719–726.

[45] P. Zhao, S. Mohan, and R. Vasudevan, "Control synthesis for nonlinear optimal control via convex relaxations," in *Proc. Amer. Control Conf.*, 2017, pp. 2654–2661.

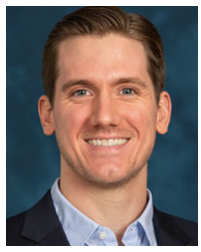
[46] T. Kalisky *et al.*, "Differential pressure control of 3D printed soft fluidic actuators," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 6207–6213.

[47] B. Tondu, "Modelling of the McKibben artificial muscle: A review," *J. Intell. Mater. Syst. Struct.*, vol. 23, no. 3, pp. 225–253, 2012.

[48] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory Implementation Applications*. Berlin, Germany: Springer, 2012.

[49] *MATLAB Version 7.10.0 (R2017a)*, MathWorks Inc., Natick, MA, USA, 2017.

[50] *Gurobi Optimizer Reference Manual*, Gurobi Optimization, LLC, Houston, TX, USA, 2018.



Daniel Bruder received the B.S. degree in engineering sciences from Harvard University, Cambridge, MA, USA, in 2013, and the M.S. and Ph.D. degrees in mechanical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2020.

He is currently a Postdoctoral Researcher with the John A. Paulson School of Engineering and Applied Sciences, Harvard University. His research interests include robotics, dynamics, and controls with a specific focus on soft robots.

Xun Fu received the B.S. degree in mechatronics engineering from Northwestern Polytechnical University, Xi'an, China, in 2018, and the master's degree in mechanical engineering in 2020 from the University of Michigan, Ann Arbor, MI, USA, where he is currently working toward the Ph.D. degree in robotics with the Robotics Institute.

His research interests include modeling, design, and control of robotic systems.

R. Brent Gillespie received the B.S. degree in mechanical engineering from the University of California, Davis, CA, USA, in 1986, the master's degree in piano performance from the San Francisco Conservatory of Music, San Francisco, CA, in 1989, and the M.S. and Ph.D. degrees in mechanical engineering from Stanford University, Stanford, CA, in 1992 and 1996, respectively.

He is currently with the Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI, USA. His current research interests include haptic interface and teleoperator control, human motor control, and robot-assisted rehabilitation after neurological injury.

C. David Remy received the Diploma degree in engineering cybernetics from the University of Stuttgart, Stuttgart, Germany, in 2007, the M.Sc. degree in mechanical engineering from the University of Wisconsin, Madison, WI, USA, in 2006, and the Ph.D. degree in science from ETH Zurich, Zürich, Switzerland, in 2011.

He is currently a Full Professor with the Institute for Nonlinear Mechanics, University of Stuttgart. His research interests include the design, simulation, and control of legged robots, exoskeletons, and other nonlinear systems. Drawing inspiration from biology and biomechanics, he is particularly interested in the effects, exploitation, and control of natural dynamic motions.

Ram Vasudevan received the B.S. degree in electrical engineering and computer sciences, the M.S. degree in electrical engineering, and the Ph.D. degree in electrical engineering from the University of California, Berkeley, CA, USA, in 2006, 2009, and 2012, respectively.

He is currently an Assistant Professor of Mechanical Engineering with the University of Michigan, Ann Arbor, MI, USA, with an appointment in the University of Michigan's Robotics Program. His research interests include the development and application of optimization and systems theory to quantify and improve human–robot interaction.