Towards a Universal Modeling and Control Framework for Soft Robots

by

Daniel K. Bruder

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2020

Doctoral Committee:

Assistant Professor Ram Vasudevan, Chair
Professor R. Brent Gillespie
Professor Sridhar Kota
Professor C. David Remy, University of Stutgartt

Daniel K. Bruder

bruderd@umich.edu

ORCID iD: 0000-0001-7683-2725

DEDICATION

To my parents, John and Patricia Bruder, thank you for your unwavering love, encouragement, and support.

ACKNOWLEDGMENTS

Earning a PhD is far from a solitary pursuit, and I have many people to thank for helping me along the way. First and foremost, I would like to thank my faculty advisors and mentors: Sridhar Kota, C. David Remy, Brent Gillespie, Talia Moore, and Ram Vasudevan.

Sridhar, you were the person who introduced me to the idea of soft robots. Your enthusiasm and passion for the concept made me believe a soft robotic future is possible and gave me the confidence to try to make it real. I have always appreciated the way you have valued and supported my passion for teaching. My graduate school application did not boast much research experience or publications, but you saw the value in my teaching background and brought me into your lab anyway. Serving as your GSI for Engin100 was one of the most fun and rewarding experiences of my time at Michigan.

David, thank you for inviting me to join the TRI soft robotics project and being such an incredible mentor to me ever since. You have always pushed my thinking, offered new perspectives on problems, and given me the most thorough and instructive feedback on my work. This has made me a better thinker, teacher, communicator, and writer. I am incredibly grateful to have had the opportunity to work with you and discover the sly sense of humor underneath your stoic exterior.

Brent, you have always been generous with your time and feedback, even before we officially began working together. It was a joy to be your GSI for ME567, and even more fun staying up late in the GGB lobby working on the TRI project with you. Thank you for always being forthcoming with your honest advice and encouragement. Whether discussing research ideas, career choices, or the best types of tea, you have always offered helpful insights and guidance.

Talia, you opened my mind to new ways of thinking about nature, robotics, and the intersection of the two. You also taught me the importance of effective visual communication. Thank you

for your willingness to look over dozens of versions of my figures, videos, and presentations, and always offer helpful suggestions and tips of how to make them better. Beyond your incredible research and mentorship, I am inspired by the ways you are pushing to make academia a more just and inclusive environment. I know your lab will flourish under your leadership, and I am grateful to have been a part of it at the beginning.

Ram, the best decision I made in my time at Michigan was asking you to be my co-advisor, and the thing I am most grateful for is that you said yes. As a mentor you are demanding without being overbearing, you are encouraging without pandering, you make yourself available but respect boundaries, and you bring a joy to your work that is contagious. Throughout my PhD, you pushed me to do my best work and made sure I was supported along the way. Thank you for taking me under your wing, and giving me the confidence to pursue my goals. One day I hope to be a professor myself, and I aspire to emulate the gold standard of teaching and mentorship that you embody.

Second, I would like to thank the legion of undergraduate and master's students I have had the pleasure of collaborating with over the last five years: Max Howarth, Yuxin Chen, Gusavo Tovar, Vivek Peri, Maggie Kohler, Michael Fischer, Enakshi Deb, Jihong Wang, and Xun Fu. I learned a lot from each of you and am excited to see where your passion takes you in the future. I want to give special thanks to Xun for his role in the work included in this dissertation. Your tireless effort and dedication made these results possible. I could not have asked for a more enthusiastic partner and collaborator, and I'm certain you will accomplish great things in your PhD.

Third, I would like to thank my friends and colleagues in the Compliant Systems Design Lab: Josh Bishop-Moser and Audrey Sedal. Josh, thank you for showing me the ropes and teaching me how to make FREEs. You were a great mentor and always seemed to have the answer before I finished thinking of the question. Audrey, the lab literally could not have run without you. I am amazed by your ability to mentor students, manage projects simultaneously, and stay creative. You are going to be a fantastic professor, and I look forward to being your colleague again someday.

Fourth, I would like to thank my friends and colleagues in ROAHM Lab: Pengcheng Zhao,

Patrick Holmes, Shreyas Kousik, Shannon Danforth, Sean Vaskov, Daphna Raz, Hannah Larson, Nils Smit-Anseeuw, Jinsun Liu, Matt Porter, Josh Mangelson, ... the list goes on and on. I loved coming into lab every day because of you. Somehow, you turned a freezing cold, windowless, fluorescently lit basement into my favorite place on campus.

Fifth, I would like to thank my family. Thank you to my extended family for all the encouragement over the years. I am blessed to be part of such a large and supportive family. Kathleen, you are the best big sister anyone could hope for. Thank you for always looking out for me and taking care of me, ever since we were kids. Mom and dad, everything I have ever accomplished is because of you. In all my pursuits, I aim to make you proud.

Sixth, I would like to thank my partner Mariama Runcie. Mari, whenever I have doubts about myself you make me feel like a million bucks. You fill me with confidence and inspire me to be the best version of myself. I love you and can't wait to start the next chapter of our lives together in Cambridge.

Lastly, I would like to thank my sources of funding. This work was supported by the Toyota Research Institute, and is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1256260 DGE. Any opinions, findings, and conclusions or recommendations expressed in this manuscript are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

FREE fiber-reinforced elastomeric enclosure

FRSA fiber-reinforced soft actuator

SVD singular value decomposition

EDMD Extended Dynamic Mode Decomposition

LTV linear time-variant

MPC model predictive control

NMPC nonlinear model predictive control

K-MPC Koopman-based model predictive control

K-NMPC Koopman-based nonlinear model predictive control

K-BMPC Koopman-based bilinear model predictive control

ABSTRACT

Traditional rigid-bodied robots are designed for speed, precision, and repeatability. These traits make them well suited for highly structured industrial environments, but poorly suited for the unstructured environments in which humans typically operate.

Soft robots are well suited for unstructured human environments because they can safely interact with delicate objects, absorb impacts without damage, and passively adapt their shape to their surroundings. This makes them ideal for applications that require safe robot-human interaction, but also presents modeling and control challenges. Unlike rigid-bodied robots, soft robots exhibit continuous deformation and coupling between structure and actuation and these behaviors are not readily captured by traditional robot modeling and control techniques except under restrictive simplifying assumptions.

The contribution of this work is a modeling and control framework tailored specifically to soft robots. It consists of two distinct modeling approaches. The first is a physics-based static modeling approach for systems of fluid-driven actuators. This approach leverages geometric relationships and conservation of energy to derive models that are simple and accurate enough to inform the design of soft robots, but not accurate enough to inform their control. The second is a data-driven dynamical modeling approach for arbitrary (soft) robotic systems. This approach leverages Koopman operator theory to construct models that are accurate and computationally efficient enough to be integrated into closed-loop optimal control schemes.

The proposed framework is applied to several real-world soft robotic systems, enabling the successful completion of control tasks such as trajectory following and manipulating objects of unknown mass. Since the framework is not robot specific, it has the potential to become the

dominant paradigm for the modeling and control of soft robots and lead to their more widespread adoption.

CHAPTER 1

Introduction

1.1 Motivation

Human technology is primarily composed of stiff and rigid materials. From skyscrapers to paperclips and everything in between, rigid structures are ubiquitous in engineered systems. This preference for rigidity manifests in our robots too. The vast majority of robots in use today are composed of rigid links actuated by strong heavy motors. They are fast and precise, but also massive and dangerous. Therefore, they are almost exclusively utilized in industrial settings where they can be physically separated from humans by cages and other barriers. Over the last several decades, these types of robots have facilitated increases in productivity and efficiency in the manufacturing sector [2], but have been of little use outside of factories.

Robots could facilitate similar improvements to our day-to-day lives, but this will require them to perform tasks within the unstructured environments inhabited by humans. In such settings, speed and precision are less important than safety and reliability, making the predominant morphology of current industrial robots poorly suited to address this challenge. For robots to operate safely and effectively alongside humans, a new paradigm in their design and control is needed.

We need not look far for inspiration. We are surrounded by systems in nature that safely and effectively operate within unstructured environments. These natural systems, designed by thousands of years of trial-and-error, utilize primarily soft materials to achieve performance capabilities that exceed the current state of the art in robotics. While human engineers prefer materials that are stiff

and rigid, nature most often opts for softness and flexibility.

Softness has some clear and well understood benefits. The most immediate benefit is the way in which softness mitigates contact with the environment. In collision, soft materials dissipate energy through deformation, making physical interactions with soft structures inherently safer than with rigid ones. Softness also facilitates more robust grasping and navigation by enabling passive conformity to the shapes of external objects and environments.

Despite the acknowledged benefits of softness, our affinity for rigid structures runs deep. For example, Robots like Boston Dynamics' Spot [3], the Fetch Mobile Manipulator [4], and others that are intended for human collaboration are all constructed from rigid materials. These robots may be smaller and lighter than their industrial forebearers, but they are similarly capable of inducing injury [5]. If softness promotes safer interactions between robots and humans, why aren't any of these popular commercial robots made of soft materials? The answer may lie in the way we are conditioned to think. Our predilection for rigidity is not merely reflected in the robots we design, it is also reflected in the way we model the physical world. For example, elementary physics classes assume that masses act as if concentrated at points and that solid bodies are perfectly rigid. Such abstractions are undoubtedly convenient, but untrue. While this way of thinking about the world is sufficient for designing and analyzing rigid things, it is hopelessly inadequate for describing things that are the soft and flexible. Years of conditioning makes our imaginations, and consequently our robots, inflexible.

The field of soft robotics has emerged to challenge the dominant paradigm. This field, which seeks to utilize and exploit soft materials to create ever more capable robots, has grown considerably over the last several years. Inspired by the way animals use soft materials to move in complex, unpredictable environments, soft robots could revolutionize emerging robotic application domains such as medicine, disaster response, and in-home human assistance [6].

Soft robots have the potential to exceed the capabilities of traditional rigid-bodied robots by exploiting the inherent benefits of softness. Softness allows them to adapt their overall shape to navigate unstructured environments, to safely interact with humans, to grasp delicate objects, and

to absorb impacts without damage [7]. Already, thousands of novel soft robotic devices such as grippers [8], crawlers [9], and swimmers [10] have been developed that exploit the flexibility of their bodies to achieve coarse behaviors such as grasping and locomotion. Despite their promise, however, soft robots have yet to be widely adopted. Unfortunately, softness introduces modeling and control challenges that have so far rendered soft robots incapable of achieving the precision needed for more useful tasks such as object manipulation.

Soft robots are fundamentally different from rigid-bodied robots in ways that demand a new set of modeling and control approaches tailored specifically to them. Soft robots do not exhibit localized deformation at discrete joints, but instead deform continuously along their bodies and have infinite degrees-of-freedom. In the absence of joints, there is no obvious choice of state variables to describe the geometry of a soft robot. As a result, it is unclear how best to describe the configuration of a soft robot with a finite set of parameters. Things are further complicated by the nonlinear properties of soft materials such as compliance, damping, and hysteresis which change a system's behavior over time. Because of these unique challenges, soft robots have yet to replace rigid-bodied robots in applications such as feeding [11], packing boxes in a warehouse [12], and other human assistive tasks where their inherent safety would be of value.

The central aim of this dissertation is to increase the capabilities of soft robots by developing a universal modeling and control framework for them. Models offer valuable insight into the behavior of robotic systems, allowing engineers to evaluate the efficacy of proposed designs before physically constructing them, or synthesize controllers that achieve desired performance specifications. For design, models are desired that can roughly predict system behavior based on a set of design parameters before the system is constructed. For control, models are desired that can accurately and efficiently predict system behavior based on a set of control inputs, and are compatible with existing model-based control techniques. It is difficult to construct models that satisfy both of these requirements simultaneously, therefore it is often useful to construct different models for the design and control of the same system. This dissertation presents methods for constructing both types of models for soft robots as well as methods for synthesizing accurate model-based

controllers from them.

1.2 State of the Art

A canonical approach to modeling soft robotic systems does not yet exist. Therefore, many different approaches have been utilized to model soft robots, depending on the intended application. These models can broadly be separated into two categories: physics-based models and data-driven models.

Physics-based models are constructed from observations of component material properties and first-principles, enabling them to make predictions about a system's behavior before the system is physically constructed. Thus, they are often used to inform the design of soft robots intended for particular tasks. For example, [13] and [14] present models for a class of soft actuators known as fiber-reinforced elastomeric enclosures (FREEs) which describe their kinematics and range of motion (respectively) in terms of design parameters such as length, diameter, and fiber angle. In [15] and [16], kinematic models for continuum manipulators are presented which describe their continuous geometry in terms of just a small set of design parameters. These models have proven useful for optimal model-based design of soft robotic manipulators [17]. However, due to the difficulty of representing infinite degrees of freedom and the nonlinear properties of soft materials, these physics-based models still fall short of completely capturing the complex behavior of soft robots. They are fundamentally limited in accuracy by the simplifying assumptions upon which they are based. For example, the popular piecewise constant curvature model [18] provides a low-dimensional description of the shape of continuum robots, but only under the assumption that bending occurs in sections of constant curvature. Other reduced-order models such as pseudo-rigid-body mechanics [19, 20], quasi-static [21, 22, 23], or simplified geometry [24, 25, 26] have been validated on real soft robotic systems, but they are only able to describe behavior in the subset of conditions over which their simplifying assumptions hold. Hence their accuracy is sufficient for informing design but still falls short of what is required to perform model-based control satisfac-

torily.

Data-driven models can more accurately capture the complex behavior of soft robots by utilizing observations of real systems. A primary benefit of data-driven modeling techniques is that a description of an input-output relationship suitable for model-based control can be obtained from system observations without explicitly defining a system state. This is especially useful for obtaining reduced-order models of soft robots that have essentially infinite-dimensional kinematics, without making simplifying physical assumptions. A potential downside of data-driven modeling is that it requires system behavior to be observed under a wide range of operating conditions, including those that may be dangerous to a robot or its surroundings. Fortunately, compared to conventional rigid-bodied robots, soft robots pose much less of a physical threat to themselves and their surroundings. It is hence possible to automatically and safely collect data under a wide range of operating conditions, making soft robots well suited for data-driven modeling approaches.

Within the class of data-driven methods, deep learning of neural networks has been the primary choice for describing the input-output behavior of soft manipulators. For instance, [27] used deep reinforcement learning to achieve open-loop position control of a soft manipulator comprised of fiber-reinforced actuators; [28] utilized a linearization of a neural network model and model predictive control to control the position of a bellows-actuated manipulator; and [29] used a combination of a recurrent neural network and supervised reinforcement learning to achieve closed-loop control of a pneumatically-driven soft manipulator. This controller was shown to compensate for disturbances such as end effector loading. While these results are promising, there are fundamental downsides to using neural network models for control. Namely, building a neural network model requires solving a nonlinear optimization problem for which global convergence may not be guaranteed [30], and its accuracy depends on the number of hidden layers, number of nodes per layer, activation function, and termination condition used during training, which must be tuned through trial and error until acceptable results are achieved. Furthermore, utilizing a neural network model at run-time is non-trivial since the control input usually appears nonlinearly within the computed model.

An alternative data-driven system identification method, which will be explored in detail in this dissertation, is based on Koopman operator theory. The Koopman operator provides a way to describe the evolution of the states of a (potentially) nonlinear dynamical system as a linear operation. This linearity is achieved by embedding the system's dynamics in a higher dimensional space of scalar valued functions. Because the Koopman operator is linear, it can be approximated via linear regression. Thus Koopman-based system identification avoids some of the undesirable features of nonlinear optimization such as the manual initialization and and tuning of training parameters.

The ability to approximate nonlinear dynamical systems globally as linear systems is valuable from a controls perspective, because it allows them to be controlled using linear control techniques. A growing body of literature is exploring ways of exploiting Koopman representations for control [31, 32, 33]. In [34], a low-dimensional analytical nonlinear system is converted to a linear system via the Koopman operator, then a linear quadratic regulator (LQR) formulation is used to construct a feedback control law. The Koopman-based linear control law is shown to be superior to those based on local linearizations of the system. In [35], a data-driven approach to identifying an approximation of the Koopman operator is presented called Extended Dynamic Mode Decomposition (EDMD). In [36], Koopman models identified via EDMD are combined with model predictive control (MPC) to achieve receding horizon control of nonlinear systems. So far, these methods have been extensively validated on simulations of dynamical systems, but rarely applied to real physical robotic systems. Notable exceptions are [37] which used a Koopman-based learning approach to control a Sphero SPRK robot and a Rethink Sawyer robot, and [38] which utilized a Koopman-based LQR controller to control a swimming fish robot. This dissertation presents the first successful adaptation and application of this theory to soft robots.

1.3 Contributions

This dissertation focuses on modeling approaches for soft robots that can inform their design or enable accurate real-time control. Each of the presented modeling approaches is validated on a real soft robotic system to demonstrate its real-world utility. This dissertation has four contributions. The first two are distinct modeling approaches, the third is a model-based control approach, and the final contribution is an extension of the previously presented modeling and control approaches to account for loading.

The first contribution is a physics-based static modeling framework for arbitrary combinations of fluid-driven actuators. Fluid-driven actuation is widely used in soft robotics, so the approach is broadly, even if not universally, applicable. This approach is applied to fiber-reinforced fluidic actuators to illustrate its ability to predict static behavior and inform the choice of design parameters. Much of this work originally appeared in [24] and [39].

The second contribution is a data-driven dynamic modeling framework based on Koopman operator theory. This approach provides a way to construct global linear, bilinear, and nonlinear realizations of the characteristically nonlinear dynamics of soft robots using standard least-squares linear regression. The approach is validated on a simulated planar 3-link arm and a real pneumatic soft robot arm. Much of this work originally appeared in [40].

The third contribution is a Koopman-based model predictive control framework that is capable of achieving accurate control of soft robots. This framework yields model predictive control optimization problems that are convex, making them computationally efficient enough to perform closed-loop control. This is exemplified by a demonstration of fully autonomous real-time control of a simulated planar 3-link arm and a real pneumatically actuated soft manipulator. Much of this work originally appeared in [41].

The fourth contribution is an extension of the Koopman-based modeling and control framework to explicitly account for external loading conditions. The framework consists of a modified system identification approach to incorporate loading conditions into a Koopman model, and a control algorithm that estimates loading conditions online. This improved framework is used to achieve

load invariant control of a simulated planar 3-link arm and the first successful demonstration of a fully autonomous pick-and-place task by a real soft manipulator arm.

Each chapter corresponds to one of the contributions outlined above and has a similar structure: A brief introduction, followed by a theoretical exposition, then one or more sections describing the theory applied to an actual system. An effort has been made to keep mathematical notation consistent across chapters, but some symbols may have been assigned more than one meaning. In such cases, the intended meaning should be apparent from context. Although each chapter has been adapted from a previously published manuscript, each chapter also contains new theory, experiments, and results. Additionally, much of the previously published content has been expanded upon and reorganized in an attempt to make for a more complete and coherent whole.

CHAPTER 2

Physics-based Modeling

Physics-based models are constructed from first-principles and are useful for the insights they provide about a proposed robotic system without requiring it to be physically constructed first. Even though most of the tasks robots are designed to perform involve dynamic movements, physics-based static models are particularly valuable due to their interpretability and computational tractability. Physics-based static models can be used to describe the relationship between actuator forces and end effector forces, determine actuator requirements, and estimate a robot's workspace under loading. The broad insights provided by such models enable roboticists to evaluate competing robot designs, which saves time and resources in the development of novel robots.

The theory of statics for rigid-bodied robots is well developed [42]. Joint torques are proportionally related to end effector forces by a Jacobian matrix whose derivation is based on the law of conservation of energy and the duality of force and velocity. However, these well-established results are not compatible with soft robotic systems since they are premised upon the assumption of rigid-bodies and localized deformation at joints. The goal of this chapter is to present an analogous Jacobian-based theory of statics for systems that exhibit continuous and distributed deformation, like soft robots. The key contribution of the proposed framework is an extension of the concept of the Jacobian to fluid-driven systems.

Although not all soft robots are fluid-driven, most are actuated by fluid-driven soft actuators that can produce forces without imposing a rigid structure [43, 44, 10, 9]. In these actuators, a pressurized fluid such as water or air creates a targeted deformation of a soft structure that encloses

a fluid-filled cavity. To achieve a desired type and direction of deformation, and not merely a homogeneous expansion, the stiffness of the soft structure is patterned in a specific way by adding reinforcing elements such as fibers, beams, or plates [45, 46, 7]. Prevalent examples of this type of actuator are bellows [47] McKibben actuators [48], and pneu-nets [49].

Due to their deformable structure, fluid-driven soft actuators differ in a fundamental way from traditional actuators. An electric motor, for example, essentially combines a kinematic constraint (the rotation axis of the motor, which is physically defined by a pair of bearings) with a force generating element (the electromagnetic forces, which create the motor torque). Since the motion of such an actuator is inherently limited to one dimension, multiple actuator stages are typically combined in *series* to achieve multiple degree of freedom (DOF) motions (Fig. 2.1a). This is the prevalent design, for example, in industrial robotic arms. In contrast, in a soft actuator, the force generating element is not supported by any physical kinematic constraints. The actuator produces a spatial force without constraining the motion to happen exclusively in the direction of this force. Because of this, soft actuators are particularly well suited to be combined in *parallel*, where the forces of the individual actuators are superimposed to generate a multi-dimensional spatial force (Fig. 2.1b). Such parallel combinations enable particularly compact designs of multi-DOF motion stages, which are prevalent in soft robotic manipulators [43, 46, 50].

This chapter introduces a generalized static model of parallel combinations of fluid-driven actuators. It is an adaptation of work originally presented in [24] and [39]. Section 2.1 presents the derivation of a static model for a single actuator and validation experiments on a real soft actuator. Section 2.3 extends this approach to arbitrary parallel combinations of fluid driven actuators and investigates how such combinations can be configured to enable effective control of multi-DOF forces.

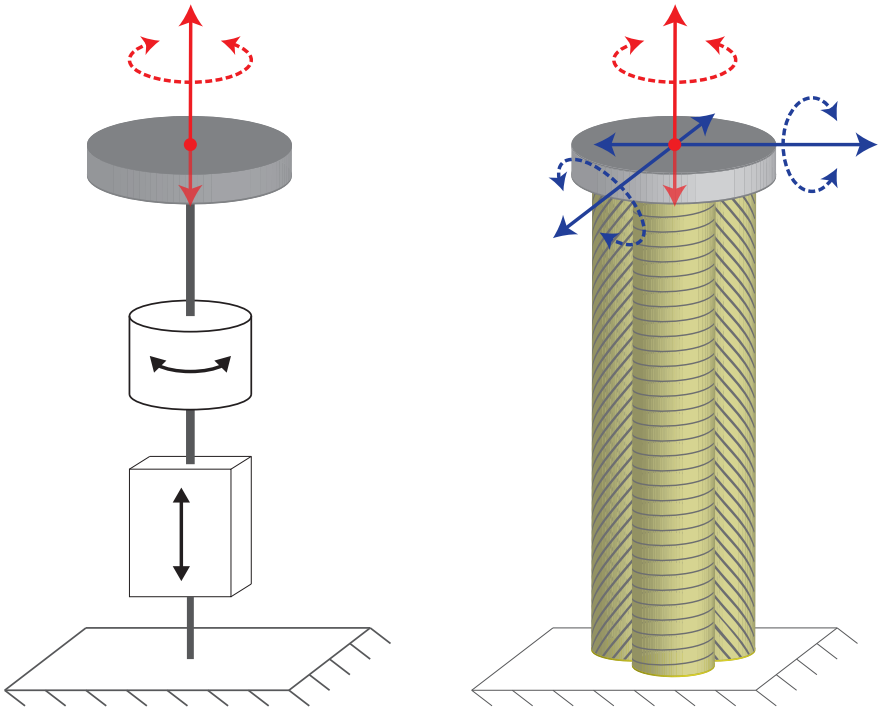


Figure 2.1: (a) A linear actuator and motor combined in *series* has the ability to generate 2-dimensional forces at the end effector (shown in red), but is constrained to motions only in the directions of these forces. (b) Three soft actuators combined in *parallel* can generate the same 2-dimensional forces at the end effector (shown in red), without imposing kinematic constraints that prohibit motion in other directions (shown in blue).

2.1 Force Generation by a Single Fluid-driven Actuator

The static modeling approach for fluid-driven robots is based on the notion of a *fluid Jacobian*, which maps the geometrical deformation of a soft actuator, or of a system of actuators, to a change in their volume. Under certain assumptions, the transpose of this Jacobian linearly maps the internal fluid pressure to the spatial forces that the actuator generates. One can think of this fluid Jacobian as the soft and multi-dimensional equivalent of the cross sectional area of a traditional pneumatic or hydraulic cylinder, since this cross section similarly relates cylinder pressure to force, and piston movement to fluid displacement.

Fluid-driven actuators contain a fluid-filled cavity and operate on the principle of converting fluid energy into mechanical work. This transmission of energy is mediated by the geometric structure of the actuator. The fluid Jacobian captures the way in which an actuator's structure constrains the volume v of the fluid cavity so that specific motions can be induced by changing the volume.

We denote by $\boldsymbol{q}$ a generalized vector of spatial deformations that fully describe the geometry of an actuator. Then, the volume of its fluid-filled cavity can be represented as a function of $\boldsymbol{q}$, i.e. $v = v(\boldsymbol{q})$. The derivative of this volume yields an expression $\dot{v}$ for the volumetric fluid flow into the actuator:

$$\dot{v}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{\partial v}{\partial t} = \frac{\partial v}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial t} = J_q(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{2.1}$$

where the fluid Jacobian is defined as $J_q = \frac{\partial v}{\partial \boldsymbol{q}}$ with respect to the deformation $\boldsymbol{q}$.

Let $\boldsymbol{\tau}_q$ denote the vector of generalized forces exerted by the actuator. It is important to note that $\boldsymbol{\tau}_q$ and $\dot{\boldsymbol{q}}$ live in dual spaces. That is, forces in $\boldsymbol{\tau}_q$ correspond to linear motion in $\dot{\boldsymbol{q}}$, and moments in $\boldsymbol{\tau}_q$ correspond to angular motion in $\dot{\boldsymbol{q}}$. In the most general case, these are 6-dimensional vectors with three translations and three rotations. In such a case, J_q is a 1×6 matrix.

Assuming that the actuator does not store energy, energy conservation dictates that the mechanical power P_{mech} generated by the actuator must equal the fluid power P_{fluid} that goes into the

actuator:

$$P_{\text{mech}} = P_{\text{fluid}} \qquad \Longrightarrow \qquad \dot{\boldsymbol{q}}^T \boldsymbol{\tau}_q = \dot{v}p, \tag{2.2}$$

where $p > 0$ is the gauge pressure of the fluid inside the actuator. Substituting (2.1) into (2.2), yields an expression for the generalized actuator force that is linear with respect to the fluid pressure:

$$\boldsymbol{\tau}_q(\boldsymbol{q}, p) = J_q^\top(\boldsymbol{q})p \tag{2.3}$$

The fluid Jacobian thus describes the directions in which a fluid-driven actuator can produce forces and moments. Since the input pressure p needs to be strictly positive to avoid collapsing of the fluid cavity, forces can only be produced in the positive direction defined by J_q.

2.2 Application: Fiber-Reinforced Elastomeric Enclosure

A particularly versatile type of fluid-driven soft actuator is the fiber-reinforced elastomeric enclosure (FREE) [13, 51, 52], also known as the fiber-reinforced soft actuator (FRSA) [45, 53, 54]. A FREE consists of a fluid-filled elastomeric tube wound with reinforcing fibers that pattern its stiffness to yield a desired mode and direction of deformation upon pressurization. By changing the angles and arrangement of these fibers, a FREE can be customized to yield a large variety of desired deformations and forces [13]. This customizability combined with their flexibility and tube-like shape makes these actuators well suited for applications such as a pipe inspection [53], catheter devices [55], or continuum manipulators [43].

One of the benefits of using FREEs as actuators is their mechanical programmability. By changing the fiber angle, Γ, of a single set of evenly distributed parallel fibers (Fig. 2.2), a FREE can be configured to produce a variety of combinations of forces along its main axis as well as twisting moments about this axis. The principle behind this adaptability can best be understood

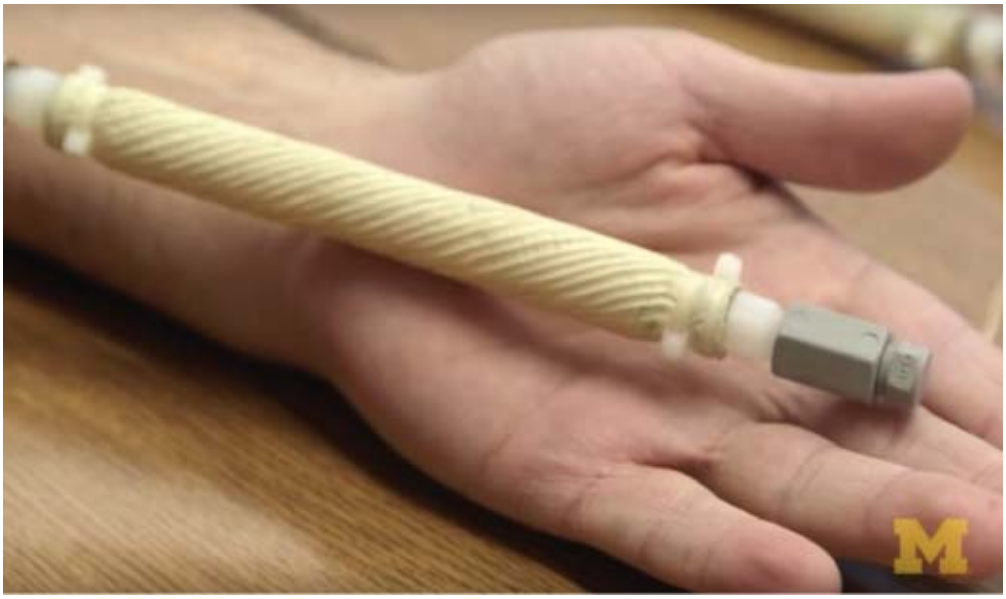


Figure 2.2: A fiber-reinforced elastomerc enclosure (FREE) is a soft fluid-driven actuator composed of an elastomer tube with fibers wound around it to impose deformations such as extension and twisting under an increase in volume.

by considering a fiber being pulled in a plane (Fig. 2.3a). For a given fiber tension, T, the ratio between the x and y components of T is determined by the angle Γ (Fig. 2.3b). In the case of a FREE, this plane is wrapped into a cylinder (Fig. 2.3c). Now the x-component of the force pulls along the direction of the central axis of the cylinder, while the y-component exerts the twisting moment. Additional effects, such as the changing radius of the FREE and the fluid pressure acting on its end-caps introduces deformation dependence [24], but the ratio between the axial force and twisting moment at the initial undeformed state is fully determined by the fiber angle Γ (Fig. 2.3d).

A number of researchers have developed ways of modeling fiber-reinforced actuators. Krishnan et. al. characterized the range of achievable motions based on fiber angles [51], exploring beyond the scope of previous literature which had focused on McKibben actuators only [48]. Bishop-Moser et. al. formalized a geometry-based kinematic model of FREEs [13], which was subsequently codified and expanded upon by others [14, 56]. Connolly et. al. took another approach, using finite element methods to predict the motions of fiber-reinforced actuators [53]. Bishop-Moser [52], Bruder [24], and Sedal [25], introduced force prediction models based upon the principle of virtual work, force balance, and continuum mechanics, respectively. Furthermore,

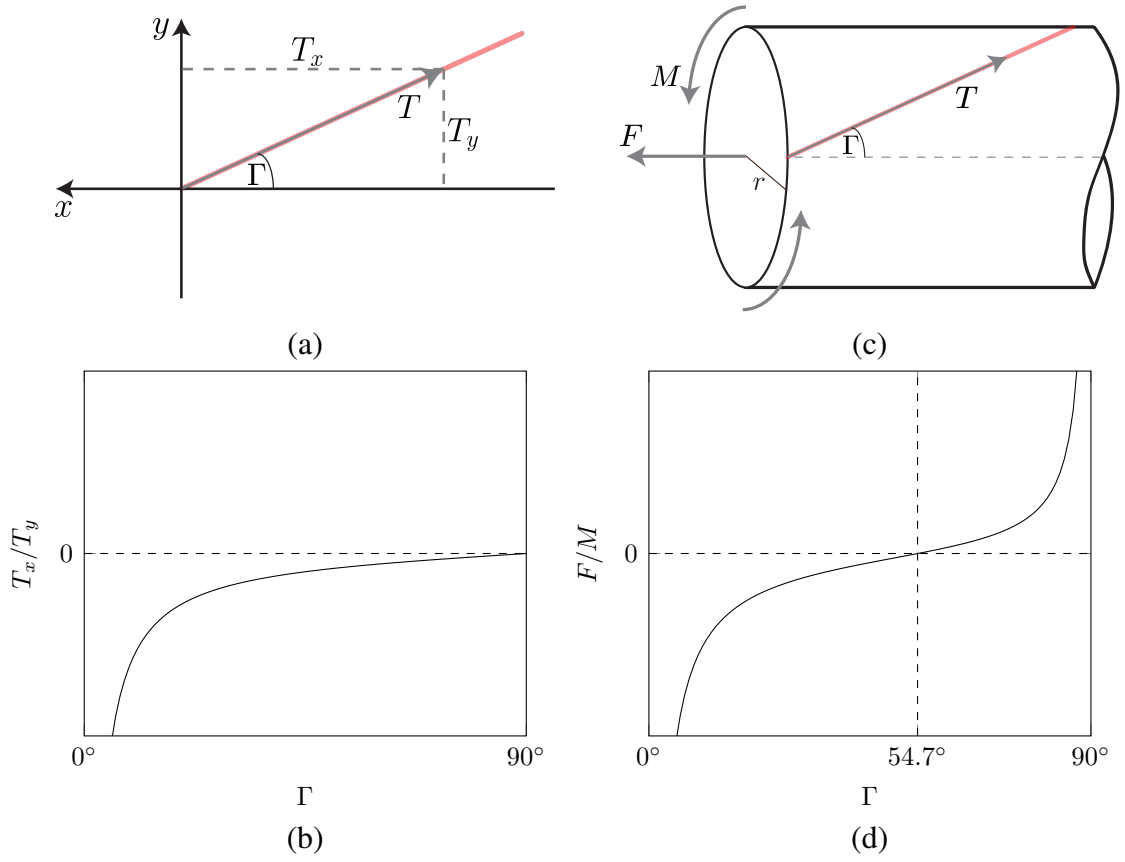


Figure 2.3: By changing the fiber angle of a FREE, it can be configured to produce a large range of force/moment combinations. To understand this, we consider how (a) the angle at which a fiber is pulled in a plane affects (b) the ratio between the x and y components of the pulling force. Similarly, by (c) wrapping the plane into a cylinder and accounting for an internal pressure force pushing out on the end-cap, the (d) ratio between axial force and twisting moment can be arbitrarily set by changing the fiber angle.

several papers have been written describing kinematic models of parallel combinations of fiber-reinforced actuators [26, 57, 58]. The distinguishing trait of the fluid Jacioan approach is that it can be applied to any fluid-driven actuators whereas these previous models apply specifically to FREEs.

To apply the fluid Jacobian approach to a FREE, we make a number of simplifying assumptions. They are consistent with those made in prior work on the modeling and control of individual FREEs [13, 24]. In particular, we assume that the fibers are inextensible and that they are uniformly distributed around an elastomeric cavity with negligible wall thickness. Under these assumptions, a FREE can be modeled as a composition of an energy transforming element (the fibers) and an energy storing element (the compliance of the elastomer body). The generalized forces generated by each of these separate elements can be superimposed to characterize the net force $\boldsymbol{\tau}_{\text{FREE}}$ produced by the FREE:

$$\boldsymbol{\tau}_{\text{FREE}} = \boldsymbol{\tau}_q + \boldsymbol{\tau}_{\text{elast}} \tag{2.4}$$

where $\boldsymbol{\tau}_q$ and $\boldsymbol{\tau}_{\text{elast}}$ are the generalized forces and torques attributed to the fiber and elastomer, respectively. The fluid Jacobian approach exclusively describes the active general forces τ_q that are generated by the fibers and that can be controlled by varying the pressure of the fluid. A supplementary elastomer model is needed to describe the elastomer force contribution.

2.2.1 Derivation of the Fluid Jacobian

The formulation of the fluid Jacobian for a FREE stems from the idea that the reinforcing fibers create a kinematic constraint on the internal volume v of the fluid cavity. Without the reinforcing fibers, this cavity could expand freely, since it is made from soft material. With the fibers, however, the volume is limited. This limitation on volume depends on the mechanical parameters of the FREE (e.g., the relaxed fiber angle or fiber length) and on the current state of geometric deformation of the FREE, represented by the generalized vecotr of spatial deformations $\boldsymbol{q}$.

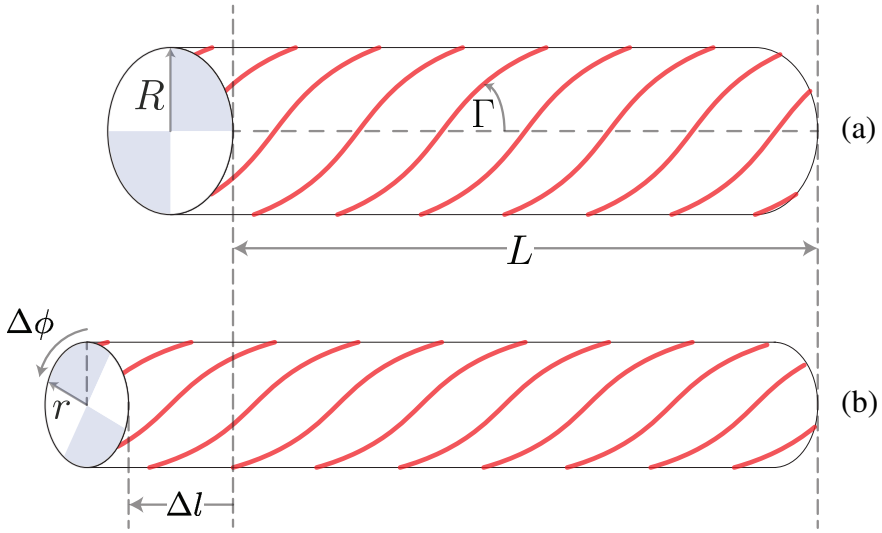


Figure 2.4: Geometric parameters of an ideal cylindrical FREE in (a) the relaxed configuration where $\boldsymbol{q} = 0$ (top), and (b) a deformed configuration where $\boldsymbol{q} = [\Delta l\ ,\ \Delta\phi]^T$ (bottom).

To make the computation of the fluid Jacobian more tractable, we rely on another common assumption for FREES: that they maintain the geometry of an ideal cylinder [13]. This neglects the tapering of a FREE towards the end-caps and any potential bending along its main axis. An ideal cylindrical FREE in its relaxed configuration (i.e. when gauge fluid pressure is zero and no external loads are applied) can be described by a set of three parameters, L, R, and Γ, where L represents the relaxed length of the FREE, R represents the radius, and Γ the fiber angle (Fig. 2.4). For notational convenience, we define two other constants from these parameters:

$$B = \left| \frac{L}{\cos\Gamma} \right| \tag{2.5}$$

$$N = -\frac{L}{2\pi R} \tan\Gamma \tag{2.6}$$

where B is the length of one of the FREE fibers, and N is the total number of revolutions the fiber makes around the FREE in the relaxed configuration.

The assumption that a FREE is cylindrical with inextensible fiber-reinforcements implies that changes in its radius, length, and twist are coupled via the right triangle relationship shown in Fig. 2.5. Therefore, its geometrical deformation can be fully defined in terms of just two parameters,

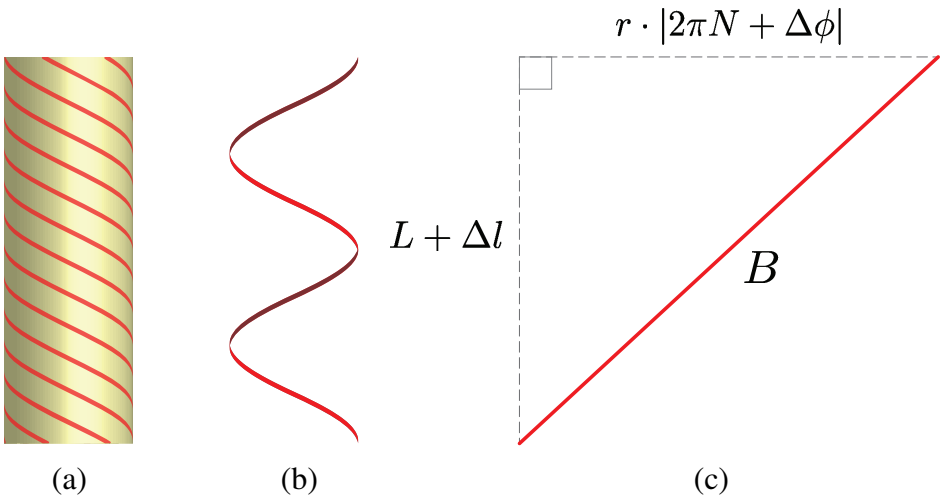


Figure 2.5: (a) A FREE contains many parallel fibers, all part of the same fiber family. (b) One isolated fiber forms a helical constraint. (c) The FREE radius r, twist $\Delta\phi$, and length change Δl are geometrically related via the right triangle formed by unwinding a fiber and laying it flat.

a change in its length Δl and a twist about its main axis $\Delta\phi$. These two values constitute the vector of generalized deformations $\boldsymbol{q} = [\Delta l, \Delta\phi]^T$. Consequently, the generalized torque vector $\boldsymbol{\tau} = [F, M]^T$ describes a force along the main axis, F, and a torsional moment about that axis, M.

Using this notation, the length l, radius r, and volume v of a deformed FREE are given by the following expressions:

$$l = L + \Delta l, \tag{2.7}$$

$$r = \frac{B}{|2\pi N + \Delta\phi|}\sqrt{1 - \left(\frac{L + \Delta l}{B}\right)^2}, \tag{2.8}$$

$$v(\boldsymbol{q}) = \pi r^2 l = \frac{\pi(L + \Delta l)(B^2 - (L + \Delta l)^2)}{(2\pi N + \Delta\phi)^2}. \tag{2.9}$$

The fluid Jacobian, which is the partial derivative of volume v with respect to deformation $\boldsymbol{q}$, then evaluates to:

$$J_q(\boldsymbol{q}) = \frac{\partial v}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\pi\left(B^2 - 3(L+\Delta l)^2\right)}{(2\pi N + \Delta\phi)^2} & \frac{2\pi(L+\Delta l)\left((L+\Delta l)^2 - B^2\right)}{(2\pi N + \Delta\phi)^3} \end{bmatrix}. \tag{2.10}$$

2.2.2 Experimental Validation

The fluid Jacobian modeling approach was applied to two FREEs with fiber angle $\Gamma = 40°$, radius $R = 5$ mm, and lengths $L = 80$ mm and 144 mm. The Jacobian for each FREE was calculated according to (2.10).

Data was collected so that model predictions could be compared to actual behavior. For each FREE, one end was fixed while the other end was free to translate and rotate. The unconstrained end of the FREE was connected to a rotary and linear encoder to measure displacement, as shown in Fig. 2.6. A fixed torsional load was applied to the unconstrained end, then the gauge pressure inside the FREE was slowly swept from 0 to 100 kPa while the displacement was recorded at various points.

As stated in the previous section, we characterize the net force generated by a FREE to be the sum of an energy transforming element and an energy storing element (2.4). The fluid Jacobian provides a method for computing the force contribution of the energy transforming element $\boldsymbol{\tau}_q$, but a model for the elastomeric force contribution $\boldsymbol{\tau}_{\text{elast}}$ is still needed. For this example, we assume that the elastomer tube acts as a coupled axial and rotational linear spring pair. This means that the elestomer force can be expressed as a function of displacement in the following form:

$$\boldsymbol{\tau}_{\text{elast}}(\boldsymbol{q}) = K_{\text{elast}}\boldsymbol{q} = \begin{bmatrix} K_{11} & K_{12} \\ K_{12} & K_{22} \end{bmatrix} \begin{bmatrix} \Delta l \\ \Delta\phi \end{bmatrix} \tag{2.11}$$

where K_{elast} is a symmetric linear stiffness matrix. The components K_{11}, K_{12}, K_{22} for a particular FREE are identified via linear regression on actual system measurements according to the following procedure:

1. Measure $\boldsymbol{q}_i = [\Delta l_i, \Delta\phi_i]^\top$ of the FREE over a set of K input pressures $\{p_i \in [0, 300]\text{kPa}\}_{i=1}^{K}$ using the apparatus shown in Fig. 2.6.

2. Compute the active force $\boldsymbol{\tau}_{q,i} = J_q^\top(\boldsymbol{q}_i)p_i$ for each $i = 1, ..., K$.

3. Set $\boldsymbol{\tau}_{\text{elast},i} = -\boldsymbol{\tau}_{q,i}$, since measurements were taken at static equilibrium and no external load

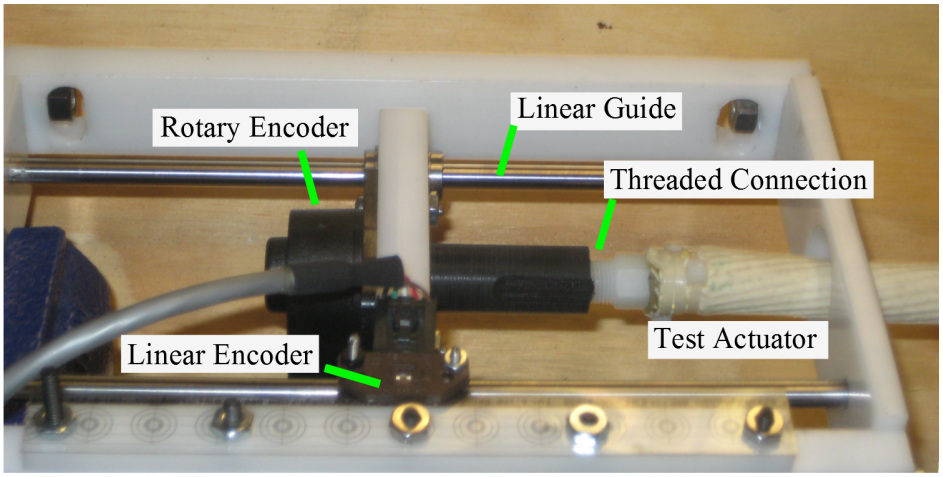


Figure 2.6: With one end of a pressurized FREE fixed in place (not shown), this sensor test setup uses linear and rotary encoders to measure the axial and rotational displacements of the other end.

was applied.

4. Define the following regression matrices:

$$T = \begin{bmatrix} \boldsymbol{\tau}_{\text{elast},1} \\ \vdots \\ \boldsymbol{\tau}_{\text{elast},K} \end{bmatrix}, \qquad Q = \begin{bmatrix} \Delta l_1 & \Delta\phi_1 & 0 \\ 0 & \Delta l_1 & \Delta\phi_1 \\ \vdots & \vdots & \vdots \\ \Delta l_K & \Delta\phi_K & 0 \\ 0 & \Delta l_K & \Delta\phi_K \end{bmatrix}, \qquad K_{\text{vec}} = \begin{bmatrix} K_{11} \\ K_{12} \\ K_{22} \end{bmatrix} \tag{2.12}$$

5. Estimate the stiffness components K_{vec} as

$$K_{\text{vec}} = Q^{\dagger}T \tag{2.13}$$

where † denotes the Moore-Penrose psuedoinverse.

We used this procedure to characterize the elastomer stiffness of our two FREEs. To evaluate the quality of the resulting model, we compare its predictions with empirical data of the measured rotation of one end of the FREE with respect to the other. All measurements were taken using the testing setup shown in Figure 2.6.

Figure 2.7 superimposes the predicted values generated by the model with those measured empirically with several torsional loads applied across various input pressures. Only model predictions for rotation are evaluated, because the FREEs demonstrated negligible changes in length. The error for each loading condition is quantified in Table 2.1.

The average model prediction error is less than 20° for all but one of the loading conditions. This level of accuracy would not be sufficient for high-precision tasks, but has been shown to enable coarser tasks such as opening a combination lock [24]. It should be noted that more accurate predictions could be acheived using the fluid Jacobian approach if it is supplemented with a more sophisticated elastomer model such as a continuum model [25, 59] or computed finite element model [53, 60].

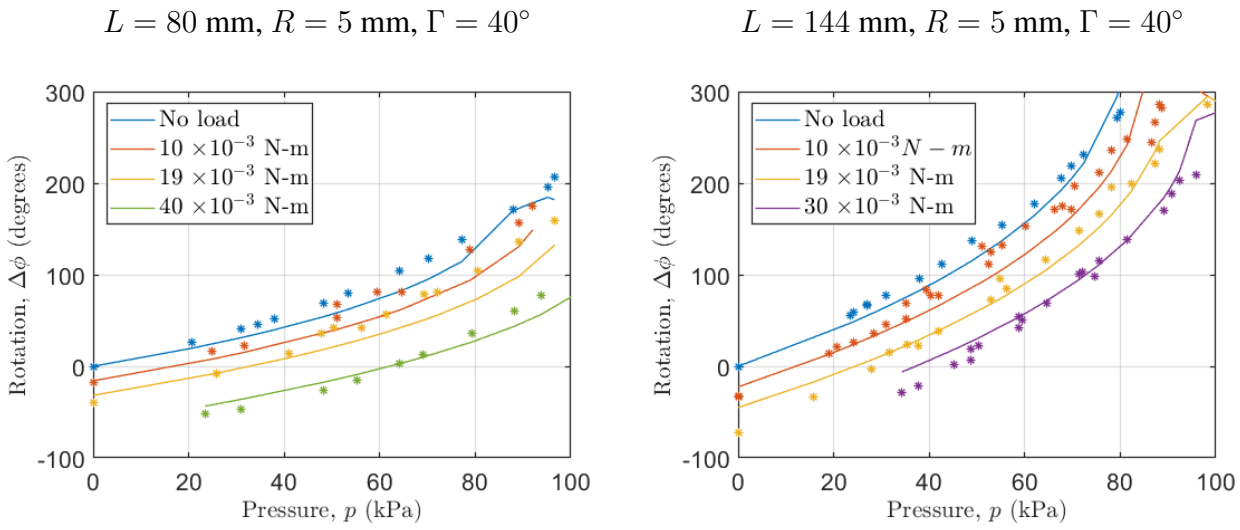


Figure 2.7: Predicted (-) and measured (*) rotation versus pressure for various torsional loads applied to a single FREE. Results for the 80 mm and 144 mm long FREEs are shown on the left and right, respectively.

Table 2.1: Model Prediction Error for Single FREEs

	Load (Nm)	**Rotation Error** (°)	
		Average	Std. Dev.
$L = 80$ mm	0×10^{-3}	13.5	8.2
	10×10^{-3}	18.9	10.8
	19×10^{-3}	19.4	11.3
	40×10^{-3}	10.6	7.3
$L = 144$ mm	0×10^{-3}	18.3	16.3
	10×10^{-3}	24.2	20.9
	19×10^{-3}	11.4	8.4
	30×10^{-3}	13.1	13.4

2.3 Force Generation by Parallel Combinations of Fluid-driven Actuators

We can extend the concept of a fluid Jacobian to systems with multiple actuators that are mounted in parallel. Actuators that are mounted in parallel each have one end attached to a common ground and the other rigidly attached to an end effector. The position and orientation of this end effector is given by a deformation vector $\boldsymbol{x}$, and we assume that an inverse kinematics function allows the computation of the state $\boldsymbol{q}_i(\boldsymbol{x})$ for each individual actuator. To express forces and moments in a common reference frame, we define a body-fixed reference point and coordinate system that are attached to the end effector (Fig. 2.8). Expressed in these coordinates, a position vector $\boldsymbol{d}_i = \begin{bmatrix} d_i^{\hat{x}_e} & d_i^{\hat{y}_e} & d_i^{\hat{z}_e} \end{bmatrix}^\top$ defines the point where the ith actuator is attached, and a unit vector $\hat{\boldsymbol{a}}_i = \begin{bmatrix} a_i^{\hat{x}_e} & a_i^{\hat{y}_e} & a_i^{\hat{z}_e} \end{bmatrix}^\top$, expresses the direction of the associated actuator axis.

Let $\boldsymbol{\tau}_{x,i}$ be the vector of generalized forces exerted by the ith actuator and expressed in end

effector coordinates:

$$\boldsymbol{\tau}_{x,i} = \begin{bmatrix} F_i^{\hat{x}_e} & F_i^{\hat{y}_e} & F_i^{\hat{z}_e} & M_i^{\hat{x}_e} & M_i^{\hat{y}_e} & M_i^{\hat{z}_e} \end{bmatrix}^\top, \tag{2.14}$$

where $F_i^{\hat{x}_e}$ is the component of force along the x-axis of the end effector frame and $M_i^{\hat{x}_e}$ is the moment about the x-axis of that frame. The components of this vector can be computed from the axial force F_i and twisting torque M_i of the ith actuator as:

$$\begin{bmatrix} F_i^{\hat{x}_e} & F_i^{\hat{y}_e} & F_i^{\hat{z}_e} \end{bmatrix}^\top = \hat{\boldsymbol{a}}_i F_i, \tag{2.15}$$

and

$$\begin{bmatrix} M_i^{\hat{x}_e} & M_i^{\hat{y}_e} & M_i^{\hat{z}_e} \end{bmatrix}^\top = \lfloor \boldsymbol{d}_i \times \rfloor \hat{\boldsymbol{a}}_i F_i + \hat{\boldsymbol{a}}_i M_i, \tag{2.16}$$

where $\lfloor \boldsymbol{d}_i \times \rfloor$ is the matrix notation for the cross-product with $\boldsymbol{d}_i$ defined as follows:

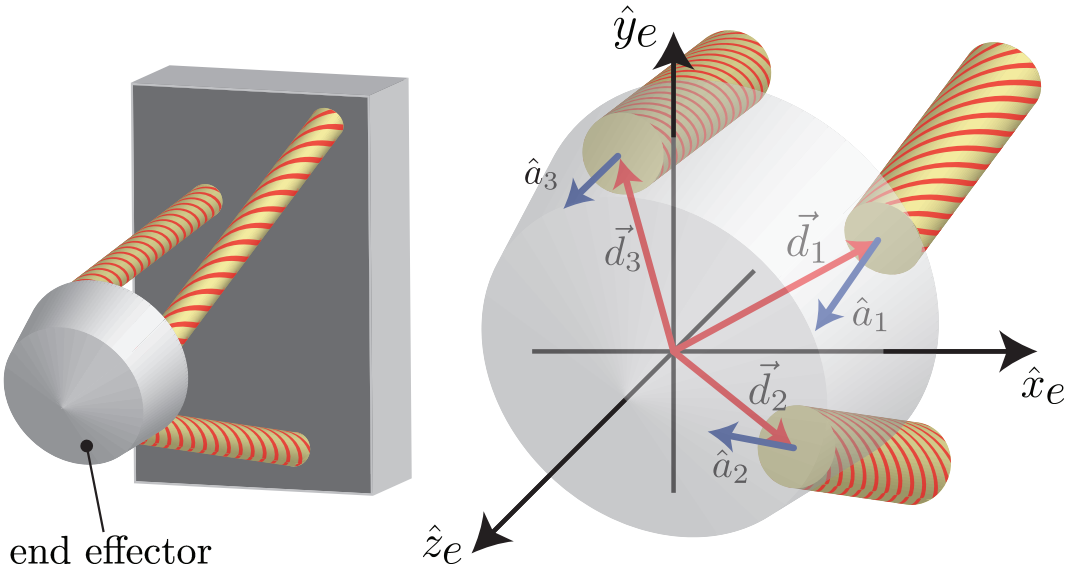
$$\lfloor \boldsymbol{d}_i \times \rfloor = \begin{bmatrix} 0 & -d_i^{\hat{z}_e} & d_i^{\hat{y}_e} \\ d_i^{\hat{z}_e} & 0 & -d_i^{\hat{x}_e} \\ -d_i^{\hat{y}_e} & d_i^{\hat{x}_e} & 0 \end{bmatrix}. \tag{2.17}$$

Combining (2.15) and (2.16) into a single transformation yields:

$$\boldsymbol{\tau}_{x,i} = \mathcal{D}_i \boldsymbol{\tau}_{q,i}, \tag{2.18}$$

where $\mathcal{D}_i$ is the 6×2 matrix defined as follows:

$$\bar{\mathcal{D}}_i = \begin{bmatrix} \begin{bmatrix} \hat{\boldsymbol{a}}_i & \mathbf{0}_{3\times 1} \end{bmatrix} \\ \begin{bmatrix} \lfloor \boldsymbol{d}_i \times \rfloor \hat{\boldsymbol{a}}_i & \mathbf{0}_{3\times 1} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times 1} & \hat{\boldsymbol{a}}_i \end{bmatrix} \end{bmatrix}. \tag{2.19}$$



(a) (b)

Figure 2.8: (a) When multiple actuators are attached in parallel to the same end effector, shown on the left, we must consider their relative positions and orientations. (b) This is determined by the vector $\boldsymbol{d}_i$ of the displacement of the attachment point of an actuator relative to the origin of the end effector frame, and by the unit vector $\hat{\boldsymbol{a}}_i$ that is aligned with the actuator axis at the attachment point, shown in the zoomed-in view on the right.

Since the actuators are mounted in parallel, the total force $\boldsymbol{\tau}_x$ is the sum of the individual forces of all n actuators connected to the end effector:

$$\boldsymbol{\tau}_x(\boldsymbol{q}, \boldsymbol{p}) = \sum_{i=1}^{n} \boldsymbol{\tau}_{x,i} = \sum_{i=1}^{n} \mathcal{D}_i J_{q,i}^{\top}(\boldsymbol{q}_i) p_i = \sum_{i=1}^{n} J_{x,i}^{\top}(\boldsymbol{x}) p_i, \tag{2.20}$$

where $J_{x,i} = J_{q,i}\mathcal{D}_i^{\top}$ is the fluid Jacobian of an individual actuator expressed in end effector coordinates and $\boldsymbol{p}$ is the vector of all actuator internal pressure values.

This can be written compactly in matrix notation as

$$\boldsymbol{\tau}_x(\boldsymbol{x}, \boldsymbol{p}) = J_x^{\top}(\boldsymbol{x})\boldsymbol{p}, \tag{2.21}$$

with the overall fluid Jacobian J_x:

$$J_x = \begin{bmatrix} J_{x,1}^\top & J_{x,2}^\top & \cdots & J_{x,n}^\top \end{bmatrix}^\top \tag{2.22}$$

Equation (2.21) shows that the force capability of the parallel combination of multiple actuators is a linear function of the pressures in the individual actuators. Since the fluid Jacobian J_x depends on the end effector state $\boldsymbol{x}$, the ability of such a system to generate forces at the end effector will vary based on its displacement. For some values of $\boldsymbol{x}$ it may be possible to generate spacial forces in multiple directions, while for others the span of possible forces may be narrower. The *force zonotope* of a parallel combination of actuators, similar to the force ellipsoid of rigid manipulators, describes the set of forces that can be generated at the end effector with a bounded set of input pressures.

Definition 2.3.1 (Force Zonotope) *For a parallel combination of n fluid-driven actuators, the force zonotope is the set of active general forces that can be generated in a specific end effector state, $\boldsymbol{x}$,*

$$\mathcal{Z}(\boldsymbol{x}) = \left\{ J_x^\top(\boldsymbol{x})\boldsymbol{p} \,:\, p_i \in [0, p_i^{max}] \right\} \tag{2.23}$$

where p_i^{max} is the maximum pressure allowed for the i^{th} actuator.

Note that the *force zonotope* is the convex hull of all spacial forces generated when $p_i \in \{0, p_i^{\max}\}$. This makes it viable to compute using a convex hull algorithm. By calculating the zonotope over a range of states, it can be used to verify that a given parallel actuator design is capable of generating the desired forces over the range of its workspace.

The force zonotope is a useful tool for design of fluid actuated systems. For example, consider the parallel combination of FREEs shown in Figure 2.9. Here, pairs of FREEs with opposite chirality are connected to the two sides of an end effector. This end effector is constrained to slide and rotate exclusively along and about its z-axis. Since the motion of the end-effector is

two-dimensional, we would like to achieve controllable forces within these two dimensions; that is, have control authority over F^{z_e} and M^{z_e}. When constructing the zonotope one FREE at time (Figure 2.9a-d), one can observe how all four FREEs are needed to achieve this control authority. In particular, it becomes evident that to achieve full control in n dimensions, at least $n+1$ individual FREEs are required in an antagonistic configuration. The additional FREE is needed because these soft actuators cannot be driven with negative pressure hence they cannot produce bidirectional forces. One can also observe that if the FREEs are chosen or arranged poorly such that the directions in J_x do not cover the space of desired forces (such as in Fig. 2.9c), this minimum number of actuators might not be sufficient.

2.4 Application: Parallel Combination of Fiber-Reinforced Elastometic Enclosures

To demonstrate the viability of the fluid Jacobian modeling methodology for parallel combinations of actuators, we show experimentally how, through the deliberate combination and configuration of parallel FREEs, full control over 2-dimensional spacial forces can be achieved by using only the minimum combination of three FREEs. To this end, we carefully chose the fiber angle Γ of each of these actuators to achieve a well-balanced force zonotope (Fig. 2.10). We combined a contracting and counterclockwise twisting FREE with a fiber angle of $\Gamma = 48°$, a contracting and clockwise twisting FREE with $\Gamma = -48°$, and an extending FREE with $\Gamma = -85°$. All three FREEs were designed with a nominal radius of R = 5 mm and a length of L = 100 mm.

2.4.1 Experimental Setup

To measure the forces generated by this actuator combination under a varying state $\boldsymbol{x}$ and pressure input $\boldsymbol{p}$, we developed a custom built test platform (Fig. 4.3). In the test platform, a linear actuator (ServoCity HDA 6-50) and a rotational servomotor (Hitec HS-645mg) were used to impose a 2-dimensional displacement on the end effector. A force load cell (LoadStar RAS1-25lb) and a

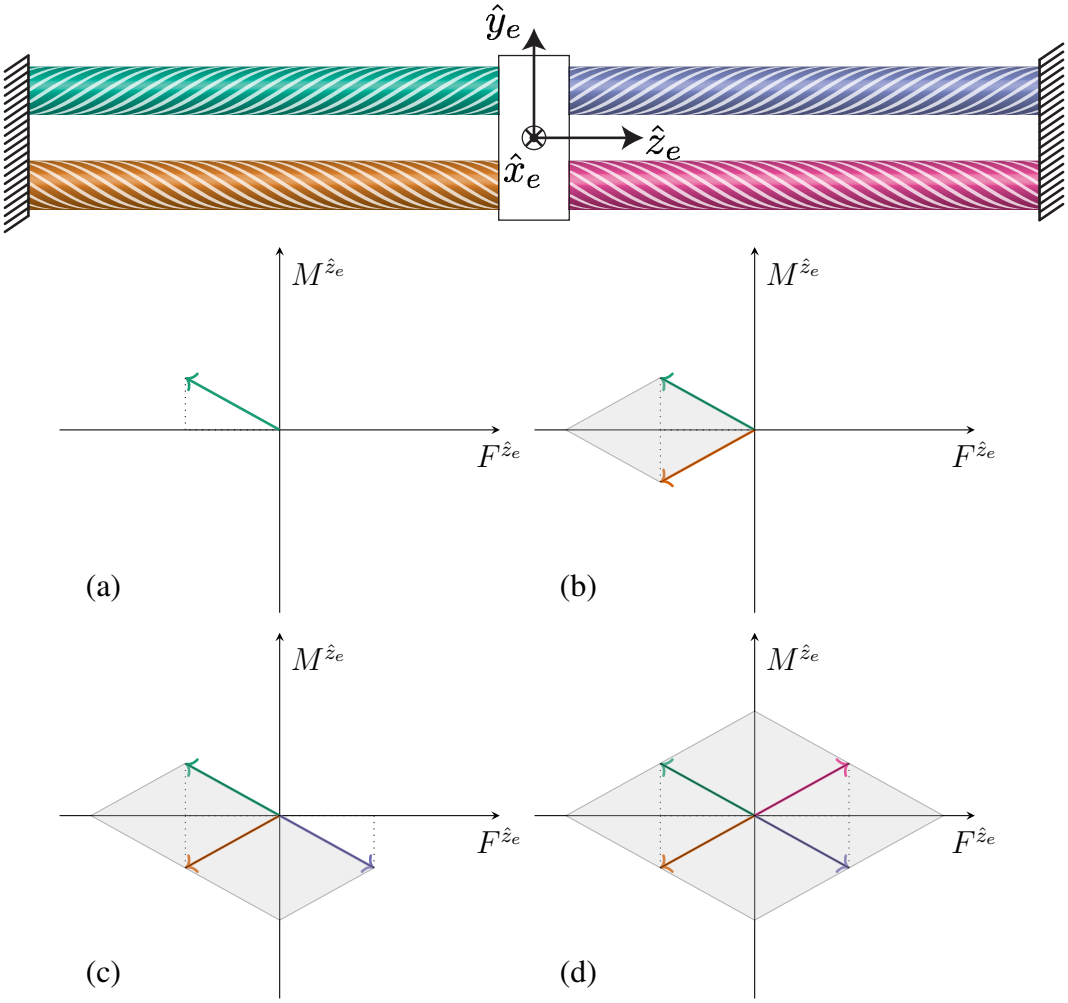


Figure 2.9: An end effector is driven by the parallel combination of two pairs of FREEs with opposing chirality. The zonotope (grey areas) is the range of forces that can be produced by applying strictly positive pressure to the individual FREEs. It is spanned by the individual force vectors that each FREE produces at maximal pressure (plotted here in the color corresponding to the FREE's appearance in the diagram above). By constructing the zonotope for (a) one FREE, (b) two FREEs, (c) three FREEs, (d) four FREEs in parallel, one can observe that all four actuators are needed to gain full control authority over forces along and torques about the z-axis. In this poorly designed system (with fiber angles and attachments points as shown in the top diagram), the theoretical minimum of 3 actuators is not sufficient to achieve full control authority.

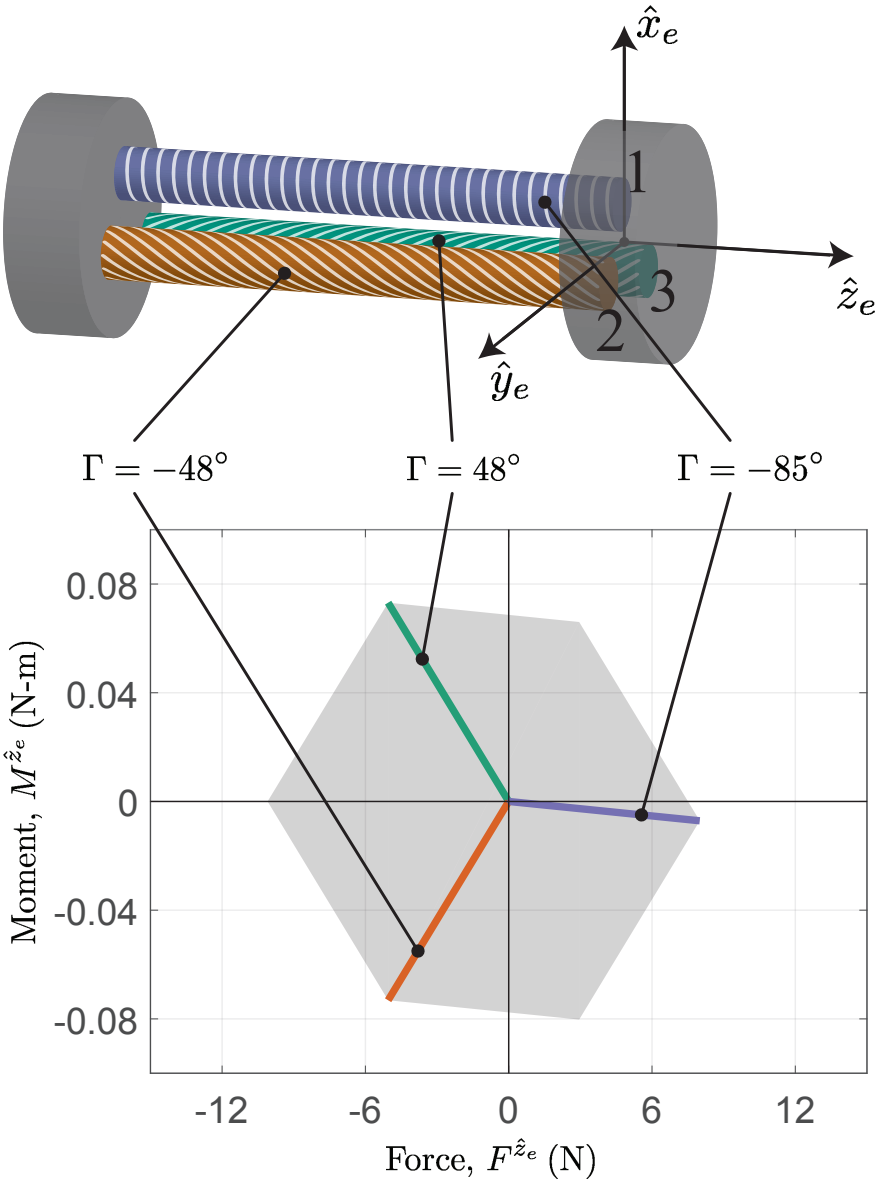


Figure 2.10: In the experimental evaluation, we employed a parallel combination of three FREEs (top) to yield forces along and moments about the z-axis of an end effector. The FREEs were carefully selected to yield a well-balanced force zonotope (bottom) to gain full control authority over $F^{\hat{z}_e}$ and $M^{\hat{z}_e}$. To this end, we used one extending FREE and two contracting FREEs which generate antagonistic moments about the end effector z-axis.

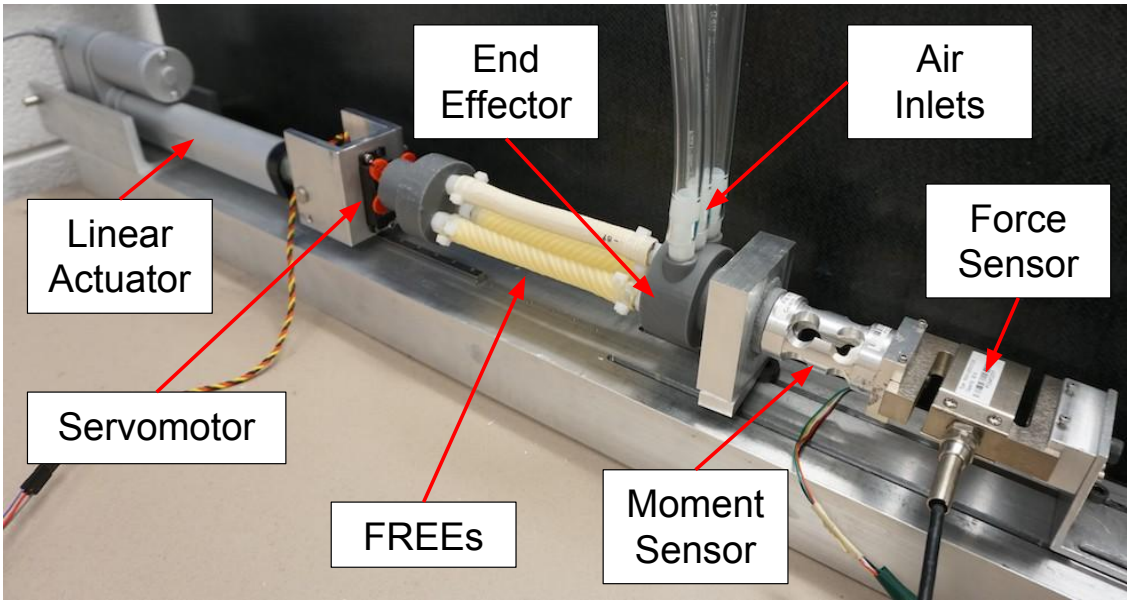


Figure 2.11: This experimental platform is used to generate a targeted displacement (extension and twist) of the end effector and to measure the forces and moments created by a parallel combination of three FREEs. A linear actuator and servomotor impose an extension and a twist, respectively, while the net force and moment generated by the FREEs is measured with a force load cell and moment load cell mounted in series.

moment load cell (LoadStar RST1-6Nm) measured the end-effector forces $F^{\hat{z}_e}$ and moments $M^{\hat{z}_e}$, respectively. During the experiments, the pressures inside the FREEs were varied using pneumatic pressure regulators (Enfield TR-010-g10-s).

The FREE attachment points (measured from the end effector origin) were measured to be:

$$\boldsymbol{d}_1 = \begin{bmatrix} 0.013 & 0 & 0 \end{bmatrix}^\top \text{m} \tag{2.24}$$

$$\boldsymbol{d}_2 = \begin{bmatrix} -0.006 & 0.011 & 0 \end{bmatrix}^\top \text{m} \tag{2.25}$$

$$\boldsymbol{d}_3 = \begin{bmatrix} -0.006 & -0.011 & 0 \end{bmatrix}^\top \text{m} \tag{2.26}$$

All three FREEs were oriented parallel to the end effector z-axis:

$$\hat{a}_i = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top, \qquad \text{for } i = 1, 2, 3 \tag{2.27}$$

Based on this geometry, the transformation matrices $\bar{\mathcal{D}}_i$ were given by:

$$\mathcal{D}_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & -0.013 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top \tag{2.28}$$

$$\mathcal{D}_2 = \begin{bmatrix} 0 & 0 & 1 & 0.011 & 0.006 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top \tag{2.29}$$

$$\mathcal{D}_3 = \begin{bmatrix} 0 & 0 & 1 & -0.011 & 0.006 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top \tag{2.30}$$

These matrices were used in equation (2.20) to yield the state-dependent fluid Jacobian J_x and to compute the resulting force zontopes.

2.4.2 Isolating the Active Force

To compare our force predictions (which focus only on the active forces induced by the fibers) to those measured empirically on the physical system, we had to remove the net elastic force due to the elastomer composition of each of the FREEs. Under the assumption that the elastomer force is merely a function of the displacement $\boldsymbol{x}$ and independent of pressure $\boldsymbol{p}$ [24], this force component can be approximated by the measured end effector force $\boldsymbol{\tau}_{x,\text{meas}}$ at a pressure of $\boldsymbol{p} = 0$. That is:

$$\boldsymbol{\tau}_{x,\text{elast}}(\boldsymbol{x}) = \boldsymbol{\tau}_{x,\,\text{meas}}(\boldsymbol{x}, 0) \tag{2.31}$$

With this, the active generalized forces were measured indirectly by subtracting off the force generated at zero pressure:

$$\boldsymbol{\tau}_x(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{\tau}_{x,\text{meas}}(\boldsymbol{x}, \boldsymbol{p}) - \boldsymbol{\tau}_{x,\text{meas}}(\boldsymbol{x}, 0) \tag{2.32}$$

2.4.3 Experimental Protocol

The force and moment generated by the parallel combination of FREEs about the end effector z-axis was measured in four different geometric configurations: neutral, extended, twisted, and simultaneously extended and twisted (see Table 2.2 for the exact deformation amounts). At each of these configurations, the forces were measured at all pressure combinations in the set

$$\mathcal{P} = \left\{ \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}^\top p^{\max} \; : \; \alpha_i = \left\{ 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1 \right\} \right\} \tag{2.33}$$

with $p^{\max}$ = 103.4 kPa. The experiment was performed twice using two different sets of FREEs to observe how fabrication variability might affect performance. The results from Trial 1 are displayed in Fig. 4.7 and the error for both trials is given by Table 2.2.

2.4.4 Results

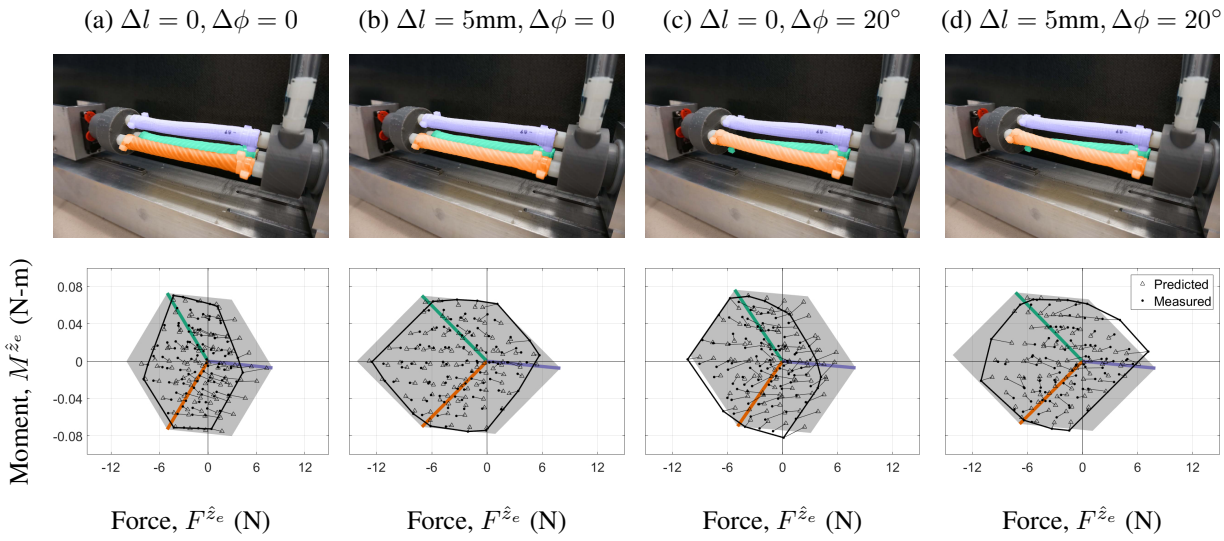


Figure 2.12: For four different deformed configurations (top row), we compare the predicted and the measured forces for the parallel combination of three FREEs (bottom row). Data points and predictions corresponding to the same input pressures are connected by a thin line, and the convex hull of the measured data points is outlined in black. The Trial 1 data is overlaid on top of the theoretical force zonotopes (grey areas) for each of the four configurations. Identical colors indicate correspondence between a FREE and its resulting force/torque direction.

For comparison, the measured forces are superimposed over the force zonotope generated by our model in Fig. 4.7a- 4.7d. To quantify the accuracy of the model, we defined the error at the j^{th} evaluation point as the difference between the modeled and measured forces

$$e_j^F = \left(F_{\text{pred},j}^{\hat{z}_e} - F_{\text{meas},j}^{\hat{z}_e} \right) \tag{2.34}$$

$$e_j^M = \left(M_{\text{pred},j}^{\hat{z}_e} - M_{\text{meas},j}^{\hat{z}_e} \right) \tag{2.35}$$

and evaluated the error across all $N = 125$ trials of a given end effector configuration. As shown in Table 2.2, the root-mean-square error (RMSE) is less than 1.5 N (8×10^{-3} Nm), and the maximum error is less than 3 N (19×10^{-3} Nm) across all trials and configurations.

Table 2.2: Root-mean-square error and maximum error

	Disp.	**RMSE**		**Max Error**	
	(mm, °)	F (N)	M (Nm)	F (N)	M (Nm)
Trial 1	0, 0	1.13	3.8×10^{-3}	2.96	7.8×10^{-3}
	5, 0	0.74	3.2×10^{-3}	2.31	7.4×10^{-3}
	0, 20	1.47	6.3×10^{-3}	2.52	15.6×10^{-3}
	5, 20	1.18	4.6×10^{-3}	2.85	12.4×10^{-3}
Trial 2	0, 0	0.93	6.0×10^{-3}	1.90	13.3×10^{-3}
	5, 0	1.00	7.7×10^{-3}	2.97	19.0×10^{-3}
	0, 20	0.77	6.9×10^{-3}	2.89	15.7×10^{-3}
	5, 20	0.95	5.3×10^{-3}	2.22	13.3×10^{-3}

It should be noted, that throughout the experiments, the FREE with a fiber angle of $-85°$ exhibited noticeable buckling behavior at pressures above ≈ 50 kPa (Fig. 2.13). This behavior was more pronounced during testing in the non-extended configurations (Fig. 4.7a and 4.7c). The buckling might explain the noticeable leftward offset of many of the points in Fig. 4.7a and Fig. 4.7c, since it is reasonable to assume that buckling reduces the efficacy of of the FREE to exert force in the

Figure 2.13: At high fluid pressure the FREE with fiber angle of $-85°$ started to buckle. This effect was less pronounced when the system was extended along the z-axis.

direction normal to the force sensor.

2.4.5 Discussion

We envision the force zonotope becoming an instrumental tool in the design of soft robotic systems. This approach could be used to choose a suitable arrangement of FREEs to actuate a compliant manipulator arm, soft orthotic device, or any other application where a custom force profile is desired. Here, we used it to systematically arrange the three FREEs in the experimental validation to ensure that the forces and torques created by the individual actuators covered all desired dimensions. Furthermore, studying the zonotope as a function of state (Fig. 2.14) allows us to make predictions about the achievable workspace of a soft robotic system. For example, the zonotope of the experimental system presented in this work will collapse for compressions of more than -10 mm. Beyond this point, it will become impossible to create contracting axial forces.

The fluid Jacobian and force zonotope models captures the physical behavior of FREEs well enough to be useful in the design of robotic applications. However, several modeling assumptions limit its accuracy and precision as they might be needed for high-fidelity control. First, FREEs are assumed to be cylindrical. This assumption introduces inaccuracy when a FREE is bent, buckled, or kinked, requiring the development of further models to predict the conditions under which these undesirable behaviors will occur. Fortunately, the fluid Jacobian approach is not inherently limited to cylindrical models and could be extended to more complicated geometrical descriptions of a

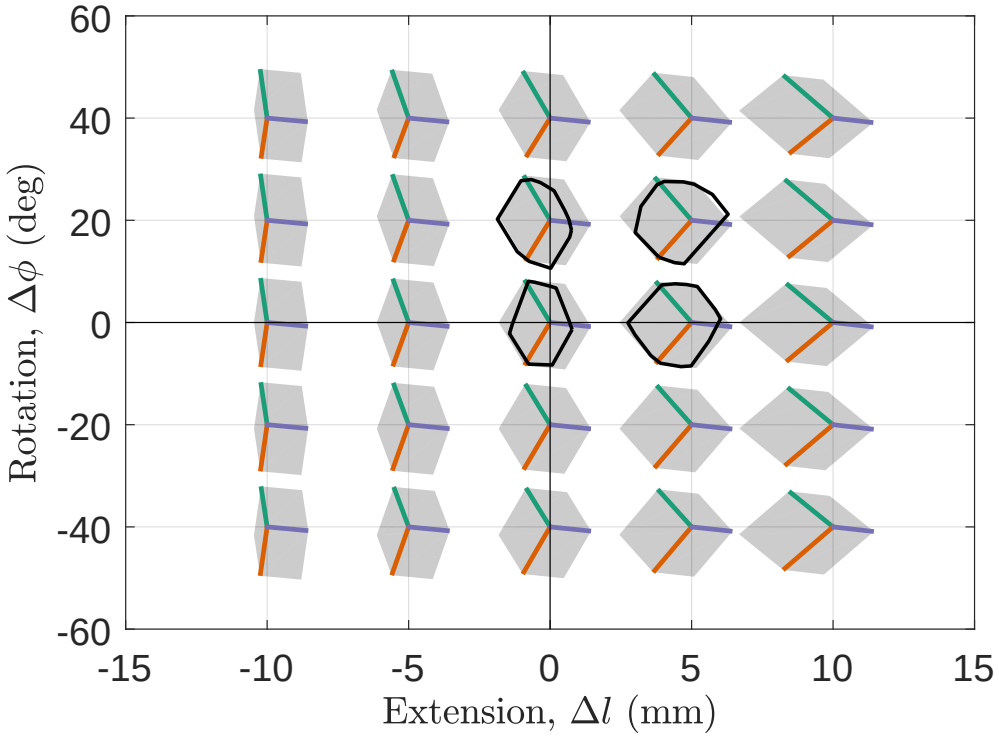


Figure 2.14: A visualization of how the *force zonotope* of the parallel combination of three FREEs (see Fig. 4.3) changes as a function of the end effector state x. One can observe that the change in the zonotope ultimately limits the work-space of such a system. In particular the zonotope will collapse for compressions of more than -10 mm. For scale and comparison, the convex hulls of the measured points from Fig. 4.7 are superimposed over their corresponding zonotope at the configurations that were evaluated experimentally.

FREE. A second, more fundamental assumption is that the model does not explicitly describe the elastomer contributions to force. That is, our approach must be supplemented with a suitable elastomer model to fully capture the force characteristics of of FREEs. While this was not the focus of the current work, such a model could be linear based on empirical data [24], derived as a continuum model from first principles [25], or computed via finite element analysis [53].

For soft robots to leverage the advantages in maneuverability, safety, and robustness of non-rigid structures, they require actuators that do not inhibit their compliance. A soft actuator such as a FREE meets this criterion because it generates a spacial force without constraining motion to occur in the direction of that force. A FREE also has the added benefit of a customizable force direction based on fiber angle; a property that was explicitly exploited in this work. Parallel combinations of FREEs can be constructed to generate arbitrary spacial forces while still retaining compliance. By characterizing the forces generated by parallel combinations of FREEs, this work lays the foundation for future applications of complex soft robotic manipulators.

CHAPTER 3

Data-driven Modeling

Recently, data-driven modeling techniques have emerged as a powerful tool to address the challenge of modeling soft robots. A primary benefit of these techniques is that a description of an input-output relationship can be obtained from system observations without explicitly defining a system state. This is especially useful for obtaining reduced-order models of continuum robots that have essentially infinite-dimensional kinematics, without making simplifying physical assumptions such as piecewise constant curvature [18], pseudo-rigid-body mechanics [19], quasi-static behavior [39, 29, 22, 23], or simplified geometry [59, 24, 25, 26]. A potential downside of data-driven modeling is that it requires system behavior to be observed under a wide range of operating conditions, including those that may be dangerous to a robot or its surroundings. Fortunately, compared to conventional rigid-bodied robots, soft robots pose much less of a physical threat to themselves and their surroundings. It is therefore possible to automatically and safely collect data under a wide range of operating conditions, making soft robots well suited for data-driven modeling approaches.

Soft robots exhibit dynamical behaviors that are distinctly nonlinear [7]. Unfortunately, identifying a nonlinear dynamical model from data typically consists of solving a nonlinear, non-convex optimization problem, for which global convergence is not guaranteed [30]. Furthermore, most nonlinear system identification methods require the manual initialization and tuning of training parameters, which have an obscure impact on the resulting model. A neural network, for example, may be able to capture the nonlinear behavior of a soft robot [61]; however, its accuracy depends

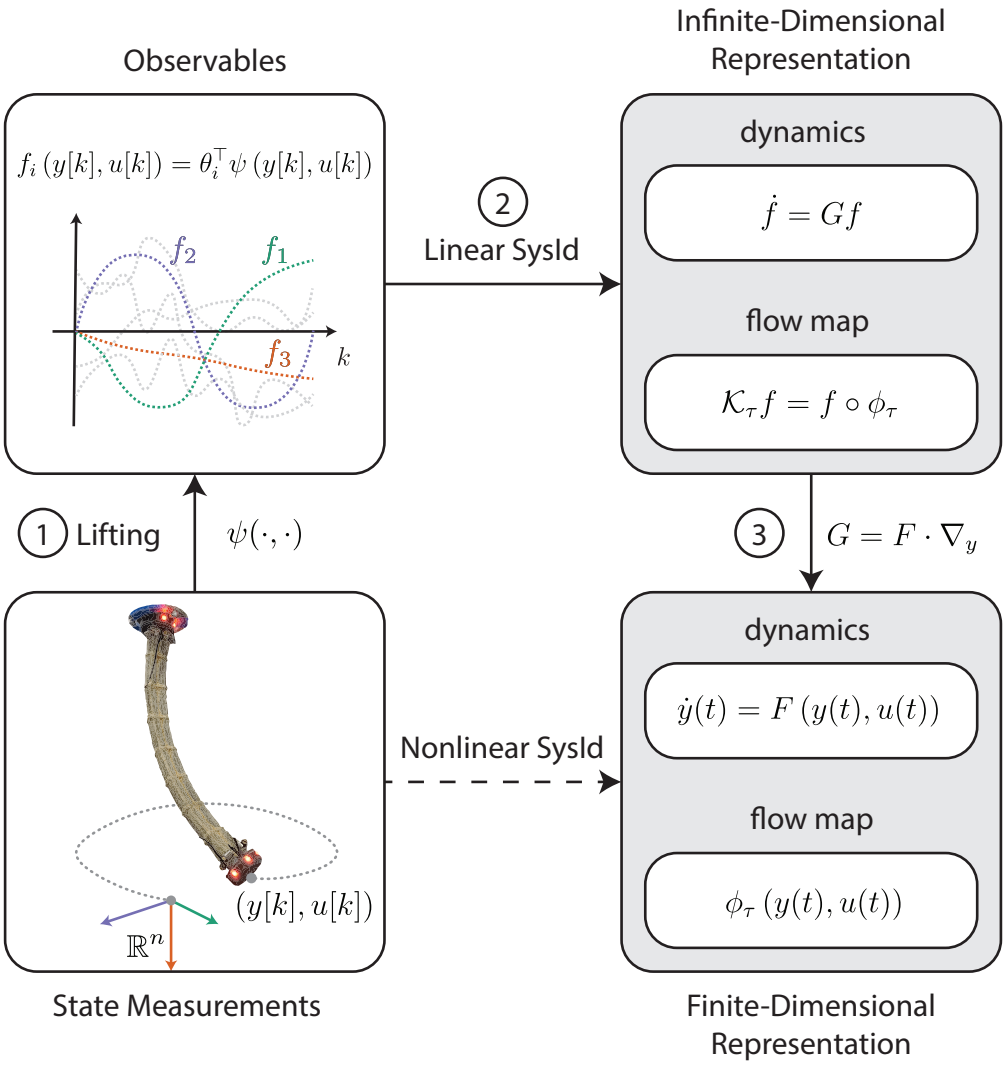


Figure 3.1: By providing an infinite-dimensional linear representation of a dynamical system, Koopman operator theory enables linear system identification of nonlinear systems. This process proceeds in three steps, as described in Section 3.1: (1) Measured states of the system are lifted to the space of real-valued functions of the state and input. (2) Least-squares regression is performed on the lifted data to obtain an approximation of the Koopman operator, $\mathcal{K}_\tau$. (3) An approximation of the nonlinear vector field $\boldsymbol{F}$ can then be obtained via its one-to-one relationship with the Koopman operator.

on the number of hidden layers, number of nodes per layer, activation function, and termination condition used during training, which must be selected through trial and error until acceptable results are achieved. Linear model identification, on the other hand, does not suffer from the typical shortcomings of nonlinear identification since linear models can be identified via linear regression [62], but linear models are poorly suited to capture the nonlinear behavior of soft robots.

This chapter describes a method to generate data-driven dynamical models of soft robots via linear regression that actually do capture their nonlinear behavior (see Fig. 3.1). This approach relies on the idea of lifting nonlinear dynamical systems to an infinite-dimensional function space where those systems have a linear representation. In this space, it is possible to describe the dynamical behavior of a system by a linear operator rather than a nonlinear vector field [32]. This linear operator, called the Koopman operator, is identified via linear regression [35], so it does not suffer from the convergence and tuning problems that are characteristic of neural networks and other nonlinear system identification methods.

Our primary contribution is demonstrating, on a real soft robotic system, a data-driven method for constructing globally valid dynamical models that does not require the manual tuning of many training parameters. To do so, we apply a Koopman-based system identification method similar to that described in [63] and [64] to create a dynamical model of a soft robot arm and verify that it captures the system's true dynamical behavior better than the models generated by several other state-of-the-art nonlinear system identification methods including a neural network, a nonlinear auto-regressive with exogenous inputs model (NLARX), a nonlinear Hammerstein-Wiener model, and a linear state space model.

The rest of this chapter is organized as follows: In Section 3.1 we formally introduce the Koopman operator and describe the system identification method. In Section 3.2 we apply the method to a simulated rigid-body system and explore the differences between several model realizations that can be constructed using the Koopman approach. In Section 3.3 we apply the method to a real soft robot system and compare the identified model's performance to that of several other models generated by various state of the art nonlinear system identification techniques. Most of the

contents of this chapter originally appeared in [40], but an attempt has been made to clarify and expand upon the original exposition of the material. Notably, this work introduces the concept of bilinear model realizations, and presents new simulation results which were absent in the original text.

3.1 Koopman-based System Identification Method

This section presents an overview of a system identification method to construct state space models of nonlinear controlled dynamical systems from input-output data. Rather than describing the evolution of a dynamical system's state directly, which may be a nonlinear mapping, the Koopman operator describes the evolution of scalar-valued functions of the state, which is a linear mapping in the infinite-dimensional space of all scalar-valued functions. A matrix approximation of this Koopman operator acting over a finite-dimensional subspace of scalar-valued function can be identified using least-squares regression on data. Then, a state space model can be constructed from the Koopman matrix. The Koopman operator is defined in Section 3.1.1, the process used to identify the Koopman matrix from data is described in Section 3.1.2, and methods for realizing linear, bilinear, and nonlinear state space models from a Koopman matrix are described in Section 3.1.3.

3.1.1 The Kooman Operator

Consider an input-output system governed by the following differential equation for the output:

$$\dot{\boldsymbol{y}}(t) = \boldsymbol{F}(\boldsymbol{y}(t), \boldsymbol{u}(t)) \tag{3.1}$$

where $\boldsymbol{y}(t) \in Y \subset \mathbb{R}^n$ is the output and $\boldsymbol{u}(t) \in U \subset \mathbb{R}^m$ is the input of the system at time $t \in [0, +\infty)$, $\boldsymbol{F}$ is a continuously differentiable function, and Y and U are compact subsets. Denote by $\phi_\tau : Y \times U \to Y \times U$ the *flow map*, where $\phi_\tau(\boldsymbol{y}(0), \boldsymbol{u}(0)) = (\boldsymbol{y}(\tau), \boldsymbol{u}(0))$ and $\boldsymbol{y}(\tau)$

is the solution to (3.1) at time τ when beginning with the initial condition $y(0)$ at time 0 and a constant input $u(0)$ applied for all time between 0 and τ.

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

$$\mathcal{K}_\tau f = f \circ \phi_\tau, \tag{3.2}$$

where $\circ$ indicates function composition. A consequence of this definition is that for a specific time step τ, the Koopman operator $\mathcal{K}_\tau$ defines an infinite-dimensional linear discrete dynamical system that advances the value of an observable by τ,

$$f(\boldsymbol{y}(t+\tau), \tilde{\boldsymbol{u}}) = \mathcal{K}_\tau f(\boldsymbol{y}(t), \tilde{\boldsymbol{u}}) \tag{3.3}$$

where $\tilde{\boldsymbol{u}}$ is a constant input over the interval $[t, t+\tau]$. Since this is true for *any* observable function f, the Koopman operator can be used to advance the output itself by applying it to the set of functions $\{f_i : f_i(\boldsymbol{y}(t), \tilde{\boldsymbol{u}}) = y_i(t)\}_{i=1}^n$ where $y_i(t)$ represents the i^{th} component of the vector $\boldsymbol{y}(t)$, advancing their values according to (3.3), and stacking the results as a vector:

$$\boldsymbol{y}(t+\tau) = \begin{bmatrix} \mathcal{K}_\tau f_1\left(\boldsymbol{y}(t), \tilde{\boldsymbol{u}}\right) \\ \vdots \\ \mathcal{K}_\tau f_n\left(\boldsymbol{y}(t), \tilde{\boldsymbol{u}}\right) \end{bmatrix}. \tag{3.4}$$

In this way, the Koopman operator provides an infinite-dimensional linear representation of a non-linear dynamical system [32]. For a specific sampling time $\tau = T_s$, (3.4) can be rewritten as a

discrete dynamical system using the following bracket notation

$$\boldsymbol{y}[i+1] = \begin{bmatrix} \mathcal{K}_{T_s} f_1 \left(\boldsymbol{y}[i], \tilde{\boldsymbol{u}} \right) \\ \vdots \\ \mathcal{K}_{T_s} f_n \left(\boldsymbol{y}[i], \tilde{\boldsymbol{u}} \right) \end{bmatrix} \tag{3.5}$$

where $i \in \mathbb{N}$ is the discrete time index and $\boldsymbol{y}[i] = \boldsymbol{y}(iT_s)$. In the remainder of this document, square brackets occurring after a time domain function will be used in this way to denote a discrete time index.

3.1.2 Identification of the Koopman Operator from Data

Since the Koopman operator is an infinite-dimensional object, we have to settle for its projection onto a finite-dimensional subspace, which can be represented as a matrix. Using the Extended Dynamic Mode Decomposition (EDMD) algorithm [35, 63, 64], we identify a finite-dimensional matrix 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 $N > n + m$ linearly independent basis functions $\{\psi_i : Y \times U \to \mathbb{R}\}_{i=1}^{N}$, and define the *lifting function* $\boldsymbol{\psi} : Y \times U \to \mathbb{R}^N$ as:

$$\psi(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) := \begin{bmatrix} \psi_1(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) & \cdots & \psi_N(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \end{bmatrix}^\top, \tag{3.6}$$

where $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$. Any observable $\bar{f} \in \bar{\mathcal{F}}$ can be expressed as a linear combination of the basis functions

$$\bar{f} = \theta_1 \psi_1 + \cdots + \theta_N \psi_N \tag{3.7}$$

where each $\theta_i \in \mathbb{R}$ is a constant. Thus $\bar{f}$ evaluated at $\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}$ can be concisely expressed as

$$\bar{f}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \boldsymbol{\theta}^\top \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \tag{3.8}$$

where $\boldsymbol{\theta} := [\theta_1 \ \cdots \ \theta_N]^\top$ acts 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 $\bar{\mathcal{K}}_\tau \in \mathbb{R}^{N \times N}$ the approximation of the Koopman operator on $\bar{\mathcal{F}}$, which operates on observables via matrix multiplication:

$$\bar{\mathcal{K}}_\tau \boldsymbol{\theta} = \boldsymbol{\theta}' \tag{3.9}$$

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

For $\bar{\mathcal{K}}_\tau$ to perfectly mimic the action of $\mathcal{K}_\tau$ on any observable $\bar{f} \in \bar{\mathcal{F}} \subset \mathcal{F}$, according to (3.2) the following should be true for all $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$,

$$\bar{\mathcal{K}}_\tau \bar{f}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \bar{f} \circ \phi_\tau(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \tag{3.10}$$

$$(\bar{\mathcal{K}}_\tau \boldsymbol{\theta})^\top \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \boldsymbol{\theta}^\top \boldsymbol{\psi} \circ \phi_\tau(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \tag{3.11}$$

$$\bar{\mathcal{K}}_\tau^\top \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \boldsymbol{\psi} \circ \phi_\tau(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}), \tag{3.12}$$

where (3.11) follows by substituting (3.8) and (3.12) follows since the result holds for all $\bar{f}$. Since this is a linear equation, it follows that for a given $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$, solving (3.12) for $\bar{\mathcal{K}}_\tau$ yields the best approximation of $\mathcal{K}_\tau$ on $\bar{\mathcal{F}}$ in the L^2-norm sense [65]:

$$\bar{\mathcal{K}}_\tau = \left(\boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}})^\top\right)^\dagger (\boldsymbol{\psi} \circ \phi_\tau(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}))^\top \tag{3.13}$$

where superscript † denotes the Moore-Penrose pseudoinverse.

To approximate the Koopman operator from a set of experimental data, we take K discrete measurements in the form of so-called "snapshots" $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}\}_{k=1}^{K}$ where

$$\boldsymbol{a}^{(k)} := \boldsymbol{y}[i^{(k)}] \tag{3.14}$$

$$\boldsymbol{b}^{(k)} := \boldsymbol{y}[i^{(k)} + 1], \tag{3.15}$$

$i^{(k)}$ denotes the discrete time index corresponding to the k^{th} measurement, $\boldsymbol{u}^{(k)}$ is the constant input applied between $a^{(k)}$ and $b^{(k)}$, and T_s is the sampling period, which is assumed to be identical for all snapshots. Note that consecutive snapshots do not have to be generated by consecutive measurements. We then lift all of the snapshots according to (3.6) and compile them into the following $K \times N$ matrices:

$$\Psi_a := \begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{a}^{(1)}, \boldsymbol{u}^{(1)})^\top \\ \vdots \\ \boldsymbol{\psi}(\boldsymbol{a}^{(K)}, \boldsymbol{u}^{(K)})^\top \end{bmatrix}, \qquad \Psi_b := \begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{b}^{(1)}, \boldsymbol{u}^{(1)})^\top \\ \vdots \\ \boldsymbol{\psi}(\boldsymbol{b}^{(K)}, \boldsymbol{u}^{(K)})^\top \end{bmatrix}. \tag{3.16}$$

$\bar{\mathcal{K}}_{T_s}$ is chosen so that it yields the least-squares best fit to all of the observed data, which, following from (3.13), is given by

$$\bar{\mathcal{K}}_{T_s} := \Psi_a^\dagger \Psi_b. \tag{3.17}$$

According to Newton's Laws, 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 to 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 these delays, we can modify the snapshots to have the following

form

$$\boldsymbol{a}^{(k)} := \begin{bmatrix} \boldsymbol{y}[i^{(k)}] \\ \vdots \\ \boldsymbol{y}[i^{(k)} - d] \\ \boldsymbol{u}[i^{(k)} - 1] \\ \vdots \\ \boldsymbol{u}[i^{(k)} - d] \end{bmatrix}, \qquad \boldsymbol{b}^{(k)} := \begin{bmatrix} \boldsymbol{y}[i^{(k)} + 1] \\ \vdots \\ \boldsymbol{y}[i^{(k)} - d + 1] \\ \boldsymbol{u}[i^{(k)} - 1] \\ \vdots \\ \boldsymbol{u}[i^{(k)} - d] \end{bmatrix} \tag{3.18}$$

where d is the number of delays. We then modify the domain of the lifting function such that $\boldsymbol{\psi} : \mathbb{R}^{\left(n+(n+m)d\right)\times m} \to \mathbb{R}^N$ to accommodate the larger dimension of the snapshots. Once these snapshots have been assembled, the Koopman matrix identification procedure is identical to the case without delays.

3.1.3 Model Realizations

The Koopman operator advances the values of real-valued functions along trajectories of a dynamical system, but does not directly describe the evolution of the system itself. This section describes how to utilize a matrix approximation of the Koopman operator to construct model realizations that describe the input-output behavior of a system.

Section 3.1.3.1 describes how to construct linear model realizations. Linear models are desirable for many reasons, primarily due to their compatibility with computationally efficient linear control design techniques. The theory for controlling linear dynamical systems is mature and there are many off-the-shelf software tools available for linear control applications. Furthermore, optimal control problems for linear systems are convex, meaning that they have unique globally optimal solutions which can often be computed quickly enough for use in real-time control schemes.

The downside of linear realizations is that they require a severe restriction on the choice of basis functions used for approximation of the Koopman operator. Specifically, no nonlinear functions of the input can be included. For some systems, this restriction fundamentally limits the accuracy of

the model, no-matter how many basis functions are included in the set.

Section 3.1.3.1 describes how to construct bilinear model realizations. Bilinear models are similar to linear models but with the addition of terms that are products of the state and input. They require a less severe restriction on the choice of basis functions, endowing them with a greater expressiveness than strictly linear models.

The downside of bilinear realizations is they are not compatible with linear control design techniques, and optimal control problems for bilinear models are not convex. However, bilinear models do still have some useful structure that can be exploited for computational efficiency. Specifically, they are linear with respect to the state for a fixed input, and linear with respect to the input for a fixed state. Thus, linear control algorithms actually can be applied indirectly to bilinear systems within a suboptimal optimization framework. Bilinear systems strike a balance between the computational efficiency of linear systems and the expressiveness of fully nonlinear systems, making them an attractive option for representing systems for which linear models are insufficient but closed-loop controllability is still desired.

Section 3.1.3.1 describes how to construct nonlinear model realizations. Nonlinear models can be constructed from arbitrary sets of basis functions, making them the most expressive of the three realizations presented. Nonlinear model realizations tend to be the most accurate, but the least computationally efficient for control applications. They are wholly incompatible with linear control design techniques, and nonlinear optimal control problems are non-convex, initialization dependent, and do not have convergence guarantees. This makes them suitable for open-loop control applications where accurate predictions are required, but not suitable for closed-loop applications.

3.1.3.1 Linear Model Realization Based on the Koopman Operator

We can use the Koopman operator to construct discrete linear models of dynamical systems with inputs of the form

$$\begin{aligned} \boldsymbol{z}[i+1] &= A\boldsymbol{z}[i] + B\boldsymbol{u}[i] \\ \boldsymbol{y}[i] &= C\boldsymbol{z}[i] \end{aligned} \tag{3.19}$$

for each $i \in \mathbb{N}$, where $\boldsymbol{y}[0] \in Y \subset \mathbb{R}^n$ is the initial output, $\boldsymbol{z}[0]$ is the initial state, and $\boldsymbol{u}[i] \in U \subset \mathbb{R}^m$ is the input applied between at the i^{th} and $(i+1)^{\text{th}}$ steps. Specifically, we desire a representation in which the input appears *linearly*, because models of this form are amenable to real-time, convex optimization techniques for feedback control design, as we describe in Section 4.1.

With a suitable choice of basis functions $\{\psi_i\}_{i=1}^N$, the Koopman matrix $\bar{\mathcal{K}}_{T_s}$ can be constructed such that it is decomposable into a linear system representation like (3.19). One way to achieve this is to define the first $N-m$ basis functions as functions of the output only, and the last m basis functions as indicator functions on each component of the input. For some $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$ we define the basis functions as,

$$\psi_i(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = g_i(\tilde{\boldsymbol{y}}), \quad \forall i \in \{1, \ldots, N-m\} \tag{3.20}$$

$$\psi_i(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \tilde{\boldsymbol{u}}_i, \quad \forall i \in \{N-m+1, \ldots, N\} \tag{3.21}$$

where $g_i : \mathbb{R}^n \to \mathbb{R}$ and $\tilde{\boldsymbol{u}}_i$ denotes the i^{th} element of $\tilde{\boldsymbol{u}}$. This choice ensures that the input only appears in the last m components of the lifted vector $\boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}})$, and an $N-m$ dimensional lifted state can be defined as $\tilde{\boldsymbol{z}} = \boldsymbol{g}(\tilde{\boldsymbol{y}}) \in \mathbb{R}^{N-m}$, where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^{N-m}$ is defined as

$$\boldsymbol{g}(\tilde{\boldsymbol{y}}) := \begin{bmatrix} g_1(\tilde{\boldsymbol{y}}) & \cdots & g_{N-m}(\tilde{\boldsymbol{y}}) \end{bmatrix}^\top. \tag{3.22}$$

Following from (3.17), the transpose of $\bar{\mathcal{K}}_{T_s}$ is the best transition matrix between the elements of the lifted snapshots in the L^2-norm sense. This implies that given the lifting functions defined in (3.20) and (3.21), $\bar{\mathcal{K}}_{T_s}$ is the minimizer to

$$\min_{\check{\mathcal{K}}} \sum_{k=1}^{K} \left\| \check{\mathcal{K}}^\top \begin{bmatrix} \boldsymbol{g}(\boldsymbol{a}^{(k)}) \\ \boldsymbol{u}^{(k)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(\boldsymbol{b}^{(k)}) \\ \boldsymbol{u}^{(k)} \end{bmatrix} \right\|_2^2. \tag{3.23}$$

Also note that given $\boldsymbol{z}[i] = \boldsymbol{g}(\boldsymbol{y}[i])$ is the state of our linear system (3.19), the best realizations of

Algorithm 1: Koopman Linear System Identification

Input: λ , $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}\}$ and $\boldsymbol{u}^{(k)}$ for $k = 1, ..., K$
Step 1: Lift data via (3.16)
Step 2: Approx. Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (3.17) or (3.48)
Step 3: Extract model matrices A, B via (3.25)
Step 4: (optional) Identify projection operator P via (3.51)
Output: A, B, $C := \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix}$,
(optional) $\hat{A} := PA$, $\hat{B} := PB$

A and B in the L^2-norm sense are the minimizers to

$$\min_{\check{A},\check{B}} \sum_{k=1}^{K} \left\| \check{A}\boldsymbol{g}(\boldsymbol{a}^{(k)}) + \check{B}\boldsymbol{u}^{(k)} - \boldsymbol{g}(\boldsymbol{b}^{(k)}) \right\|_2^2. \tag{3.24}$$

By comparing (3.23) and (3.24), one can confirm that A and B are embedded in $\bar{\mathcal{K}}_{T_s}$ and can be isolated by partitioning it as follows:

$$\bar{\mathcal{K}}_{T_s}^{\top} = \begin{bmatrix} A_{\{(N-m)\times(N-m)\}} & B_{\{(N-m)\times m\}} \\ O_{\{m\times(N-m)\}} & I_{\{m\times m\}} \end{bmatrix} \tag{3.25}$$

where I denotes an identity matrix, O denotes a zero matrix, and the subscripts in curly brackets denote the dimensions of each matrix.

For convenience, we can define the first n basis functions as indicator functions on each component of the output, i.e.

$$g_i(\tilde{\boldsymbol{y}}) = \tilde{\boldsymbol{y}}_i \quad \forall i \in \{1, \ldots, n\}. \tag{3.26}$$

Then, the output matrix C is defined as the matrix which projects the first n elements of the state onto the output-space,

$$C = \begin{bmatrix} I_{\{n\times n\}} & O_{\{n\times(N-n)\}} \end{bmatrix}. \tag{3.27}$$

3.1.3.2 Bilinear Model Realization Based on the Koopman Operator

We can use the Koopman operator to construct discrete bilinear models of dynamical systems with inputs of the form

$$\begin{aligned} \boldsymbol{z}[i+1] &= A\boldsymbol{z}[i] + B\boldsymbol{u}[i] + \sum_{j=1}^{m} H_j \boldsymbol{z}[i] u_j[i] \\ \boldsymbol{y}[i] &= C\boldsymbol{z}[i]. \end{aligned} \tag{3.28}$$

for each $i \in \mathbb{N}$, where $\boldsymbol{y}[0]n$ is the initial output, $\boldsymbol{z}[0]$ is the initial state, and $u_j[i]$ denotes the j^{th} component of the input applied between at the i^{th} and $(i+1)^{\text{th}}$ steps. This model is simultaneously linear with respect to the input and linear with respect to the state, but it includes terms that are products of the state and input.

With a suitable choice of basis functions $\{\psi_i\}_{i=1}^{N(m+1)+m}$, the Koopman matrix $\bar{\mathcal{K}}_{T_s}$ can be constructed such that it is decomposable into a bilinear system representation like (3.28). The first N basis functions $\{g_i : \mathbb{R}^n \to \mathbb{R}\}_{i=1}^{N}$ are defined as functions of the output only, the next Nm basis functions are defined as the product of the first N basis functions and each component of the input, and the last m basis functions are defined as indicator functions on each component of the input. For some $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$ the basis functions have the following form,

$$\psi_i(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \begin{cases} g_i(\tilde{\boldsymbol{y}}) & i \in \{1, ..., N\} \\ g_{i-N}(\tilde{\boldsymbol{y}})\, \tilde{u}_1 & i \in \{N+1, ..., 2N\} \\ \quad\vdots & \\ g_{i-Nm}(\tilde{\boldsymbol{y}})\, \tilde{u}_m & i \in \{Nm+1, ..., N(m+1)\} \\ \tilde{\boldsymbol{u}}_{i-N(m+1)} & i \in \{N(m+1)+1, ..., N(m+1)+m\} \end{cases} \tag{3.29}$$

We desire the lifted state to be defined in terms of the system outputs only. Thus, we define the lifted state $\tilde{\boldsymbol{z}}$ corresponding to an output $\tilde{\boldsymbol{y}}$ to be the vector composed of the first N basis functions evaluated at $\tilde{\boldsymbol{y}}$, i.e. $\tilde{\boldsymbol{z}} = \boldsymbol{g}(\tilde{\boldsymbol{y}}) \in \mathbb{R}^N$, where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^N$ is defined as the vector concatenation of

the first N basis functions,

$$\boldsymbol{g}(\tilde{\boldsymbol{y}}) \coloneqq \begin{bmatrix} g_1(\tilde{\boldsymbol{y}}) & \cdots & g_N(\tilde{\boldsymbol{y}}) \end{bmatrix}^\top. \tag{3.30}$$

Following from (3.17), the transpose of $\bar{\mathcal{K}}_{T_s}$ is the best transition matrix between the elements of the lifted snapshots in the L^2-norm sense. This implies that given the lifting functions defined in (3.29), $\bar{\mathcal{K}}_{T_s}$ is the minimizer to

$$\min_{\check{\mathcal{K}}} \sum_{k=1}^{K} \left\| \check{\mathcal{K}}^\top \begin{bmatrix} \boldsymbol{g}(\boldsymbol{a}^{(k)}) \\ \boldsymbol{g}(\boldsymbol{a}^{(k)}) u_1^{(k)} \\ \vdots \\ \boldsymbol{g}(\boldsymbol{a}^{(k)}) u_m^{(k)} \\ \boldsymbol{u}^{(k)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{g}(\boldsymbol{b}^{(k)}) \\ \boldsymbol{g}(\boldsymbol{b}^{(k)}) u_1^{(k)} \\ \vdots \\ \boldsymbol{g}(\boldsymbol{b}^{(k)}) u_m^{(k)} \\ \boldsymbol{u}^{(k)} \end{bmatrix} \right\|_2^2 \tag{3.31}$$

Also note that given $\boldsymbol{z}[i] = \boldsymbol{g}(\boldsymbol{y}[i])$ is the state of our bilinear system (3.28), the best realizations of A , B , and $\{H_j\}_{j=1}^m$ in the L^2-norm sense are the minimizers to

$$\min_{\check{A},\check{B},\{\check{H}_j\}_{j=1}^m} \sum_{k=1}^{K} \left\| \check{A}\boldsymbol{g}(\boldsymbol{a}^{(k)}) + \check{B}\boldsymbol{u}^{(k)} + \sum_{j=1}^{m} \check{H}_j \boldsymbol{g}(\boldsymbol{a}^{(k)}) u_j^{(k)} - \boldsymbol{g}(\boldsymbol{b}^{(k)}) \right\|_2^2 \tag{3.32}$$

By comparing (3.31) and (3.32), one can confirm that A , B , and $\{H_j\}_{j=1}^m$ are embedded in $\bar{\mathcal{K}}_{T_s}$ and can be isolated by partitioning it as follows:

$$\bar{\mathcal{K}}_{T_s}^\top = \begin{bmatrix} A_{\{N\times N\}} & H_{1\,\{N\times N\}} & \cdots & H_{m\,\{N\times N\}} & B_{\{N\times m\}} \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{bmatrix} \tag{3.33}$$

where the subscripts in curly brackets denote the dimensions of each matrix.

Just as with the linear model realization, we can simplify the projection from the lifted state to the output by defining the first n basis functions as indicator functions on each component of the

Algorithm 2: Koopman Bilinear System Identification

Input: λ , $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}\}$ and $\boldsymbol{u}^{(k)}$ for $k = 1, ..., K$
Step 1: Lift data via (3.16)
Step 2: Approx. Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (3.17) or (3.48)
Step 3: Extract model matrices $A, B, \{H_j\}_{j=1}^m$ via (3.33)
Step 4: (optional) Identify projection operator P via (3.51)
Output: A, B, $\{H_j\}_{j=1}^m$, $C := \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix}$,
(optional) $\hat{A} := PA$, $\hat{B} := PB$, $\{\hat{H}_j\}_{j=1}^m := \{PH_j\}_{j=1}^m$

output, i.e.

$$g_i(\tilde{\boldsymbol{y}}) = \tilde{\boldsymbol{y}}_i \quad \forall i \in \{1, \ldots, n\}. \tag{3.34}$$

Then, C is defined as the matrix which projects the first n elements of the state onto the output-space,

$$C = \begin{bmatrix} I_{\{n\times n\}} & O_{\{n\times(N-n)\}} \end{bmatrix}. \tag{3.35}$$

3.1.3.3 Discrete Nonlinear Model Realization

We can use the Koopman operator to construct discrete nonlinear models of dynamical systems with inputs of the form

$$\boldsymbol{y}[i+1] = \boldsymbol{T}(\boldsymbol{y}[i], \boldsymbol{u}[i]) \tag{3.36}$$

for each $i \in \mathbb{N}$, where $\boldsymbol{T} : Y \times U \to \mathbb{R}^n$ is a (nonlinear) discrete map, $\boldsymbol{y}[0]$ is the initial output and $\boldsymbol{u}[i]$ is the input applied between the i^{th} and $(i+1)^{\text{th}}$ steps.

This representation admits nonlinear input terms, so we allow all of the basis functions to depend on both the input and output $\{\psi_i : Y \times U \to \mathbb{R}\}_{i=1}^N$, except for the first n basis functions

which are specified to be indicator functions on each component of the output, i.e.

$$\psi_i(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \tilde{y}_i \quad \forall i \in \{1, ..., n\} \tag{3.37}$$

for some $\tilde{\boldsymbol{y}} \in Y$. Hence, the lifting function $\boldsymbol{\psi} : Y \times U \to \mathbb{R}^N$ is defined as

$$\boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \begin{bmatrix} \tilde{\boldsymbol{y}} \\ \psi_{n+1}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \\ \vdots \\ \psi_N(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \end{bmatrix} \tag{3.38}$$

for some $\tilde{\boldsymbol{y}} \in Y$ and $\tilde{\boldsymbol{u}} \in U$.

Following from (3.17), the transpose of $\bar{\mathcal{K}}_{T_s}$ is the best transition matrix between the elements of the lifted snapshots in the L^2-norm sense. This implies that $\bar{\mathcal{K}}_{T_s}$ is the minimizer to

$$\min_{\check{\mathcal{K}}} \sum_{k=1}^{K} \left\| \check{\mathcal{K}}^\top \begin{bmatrix} \boldsymbol{a}^{(k)} \\ \boldsymbol{\psi}_{n+1}(\boldsymbol{a}^{(k)}, \boldsymbol{u}^{(k)}) \\ \vdots \\ \boldsymbol{\psi}_N(\boldsymbol{a}^{(k)}, \boldsymbol{u}^{(k)}) \end{bmatrix} - \begin{bmatrix} \boldsymbol{b}^{(k)} \\ \boldsymbol{\psi}_{n+1}(\boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}) \\ \vdots \\ \boldsymbol{\psi}_N(\boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}) \end{bmatrix} \right\|_2^2 \tag{3.39}$$

Therefore, the discrete map $\boldsymbol{T} : Y \times U \to \mathbb{R}^n$ that best describes the transition from $\boldsymbol{a}^{(k)}$ to $\boldsymbol{b}^{(k)}$ in the L^2-norm sense over all k can be constructed by isolating the first n rows of the transpose of the Koopman matrix $\bar{\mathcal{K}}_{T_s}$ and multiplying by the lifting function $\boldsymbol{\psi}$,

$$\boldsymbol{T}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \begin{bmatrix} I_{\{n \times n\}} & O_{\{n \times (N-n)\}} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^\top \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \tag{3.40}$$

where I denotes an identity matrix, O denotes a zero matrix, and the subscripts in curly brackets denote the dimensions of each matrix. Algorithm 3 summarizes this discrete nonlinear system identification process.

Algorithm 3: Koopman Nonlinear System Identification (Discrete)

Input: λ , $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}\}$ and $\boldsymbol{u}^{(k)}$ for $k = 1, ..., K$
Step 1: Lift data via (3.16)
Step 2: Approx. Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (3.17) or (3.48)
Step 3: Identify discrete map $\boldsymbol{T}$ by (3.40)
Output: $\boldsymbol{T} : Y \times U \to \mathbb{R}^n$

3.1.3.4 Continuous Nonlinear Model Realization

So far we have only introduced methods for constructing discrete model realizations from a matrix approximation of the Koopman operator, but a Koopman matrix can also be used to identify a continuous nonlinear dynamical model of the form

$$\dot{\boldsymbol{y}}(t) = \boldsymbol{F}(\boldsymbol{y}(t), \boldsymbol{u}(t)) \tag{3.41}$$

for $t \geq 0$, where $\boldsymbol{F} : Y \times U \to \mathbb{R}^n$ is a (nonlinear) continuous map, $\boldsymbol{y}(0)$ is the initial output and $\boldsymbol{u}(t)$ is the input applied at time t. Using the same lifting function (3.38) defined in the previous section, the Koopman operator is once again approximated using least-squares regression via (3.16) and (3.17).

To construct a model in the form of (3.41), we introduce the infinitesimal generator of the Koopman operator $G : \mathcal{F} \to \mathcal{F}$ [66, Equation 7.6.5], which is defined in terms of the vector field $\boldsymbol{F}$ as,

$$G = \boldsymbol{F} \cdot \nabla_y \tag{3.42}$$

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 $\mathcal{K}_t$ 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:

$$\mathcal{K}_t = e^{Gt} = \sum_{k=0}^{\infty} \frac{t^k}{k!} G^k \tag{3.43}$$

With an approximation of the Koopman operator $\bar{\mathcal{K}}_{T_s}$ in hand, we can solve for its infinitesimal generator $\bar{G}$ by inverting (3.43):

$$\bar{G} = \frac{1}{T_s} \log \bar{\mathcal{K}}_{T_s} \in \mathbb{R}^{N \times N} \tag{3.44}$$

where $\log$ denotes the principal matrix logarithm [67, Chapter 11]. Note that the principal matrix logarithm is only defined for matrices whose eigenvalues all have non-negative real parts, and that $\bar{\mathcal{K}}_{T_s}$ may have zero or negative eigenvalues when the number of data points is too small [63]. 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, (3.42) can be used to identify $\boldsymbol{F}$. Consider G applied to an observable $\bar{f} \in \bar{\mathcal{F}}$ at some point $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) \in Y \times U$. According to (3.42), this is equivalent to the inner product of the vector field $\boldsymbol{F}$ and the gradient of $\bar{f}$ with respect to $\boldsymbol{y}$:

$$\bar{G}\bar{f}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \frac{\partial \bar{f}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}})}{\partial \boldsymbol{y}} \boldsymbol{F}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}). \tag{3.45}$$

Using $\boldsymbol{\theta}$ to denote the vector representation of $\bar{f}$ from (3.8) yields the following equivalent expression,

$$(\bar{G}\boldsymbol{\theta})^T \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}) = \boldsymbol{\theta}^T \frac{\partial \boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}})}{\partial \boldsymbol{y}} \bar{\boldsymbol{F}}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}). \tag{3.46}$$

We seek the vector field $\boldsymbol{F}$ such that (3.46) holds as well as possible in the L^2-norm sense for all observed data. Therefore, we choose the least-squares solution to (3.46) over the set of all

Algorithm 4: Koopman Nonlinear System Identification (Continuous)

Input: λ , $\{a^{(k)}, b^{(k)}\}$ and $u^{(k)}$ for $k = 1, ..., K$
Step 1: Lift data via (3.6)
Step 2: Approx. Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (3.17) or (3.48)
Step 3: Identify infinitesimal generator $\bar{G}$ by (3.46)
Step 4: Solve for vector field $\boldsymbol{F}$ via (3.47)
Output: $\boldsymbol{F} : Y \times U \to \mathbb{R}^n$

snapshots, which is given by

$$\boldsymbol{F} = \begin{bmatrix} \frac{\partial \boldsymbol{\psi}(\boldsymbol{a}^{(1)}, \boldsymbol{u}^{(1)})}{\partial \boldsymbol{y}} \\ \vdots \\ \frac{\partial \boldsymbol{\psi}(\boldsymbol{a}^{(K)}, \boldsymbol{u}^{(K)})}{\partial \boldsymbol{y}} \end{bmatrix}^{\dagger} \begin{bmatrix} \bar{G}^T \\ \vdots \\ \bar{G}^T \end{bmatrix} \boldsymbol{\psi}. \tag{3.47}$$

Algorithm 4 summarizes this continuous nonlinear system identification process. For a more thorough treatment, see [63, 64].

3.1.4 Practical Considerations

3.1.4.1 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 [68]. To guard against overfitting to noise while identifying $\bar{\mathcal{K}}_{T_s}$, we utilize the L^1-regularization method of Least Absolute Shrinkage and Selection Operator (LASSO) [69]:

$$\vec{\mathcal{K}}_{T_s} = \underset{\vec{\mathcal{K}}_{T_s}}{\arg\min} \; ||\vec{\Psi}_a \vec{\mathcal{K}}_{T_s} - \vec{\Psi}_b||_2^2 + \lambda ||\vec{\mathcal{K}}_{T_s}||_1 \tag{3.48}$$

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 $\lambda = 0$, (3.48) provides the same unique least-squares solution as (3.17); as λ increases it drives the elements of $\hat{\mathcal{K}}_{T_s}$ to zero. For an

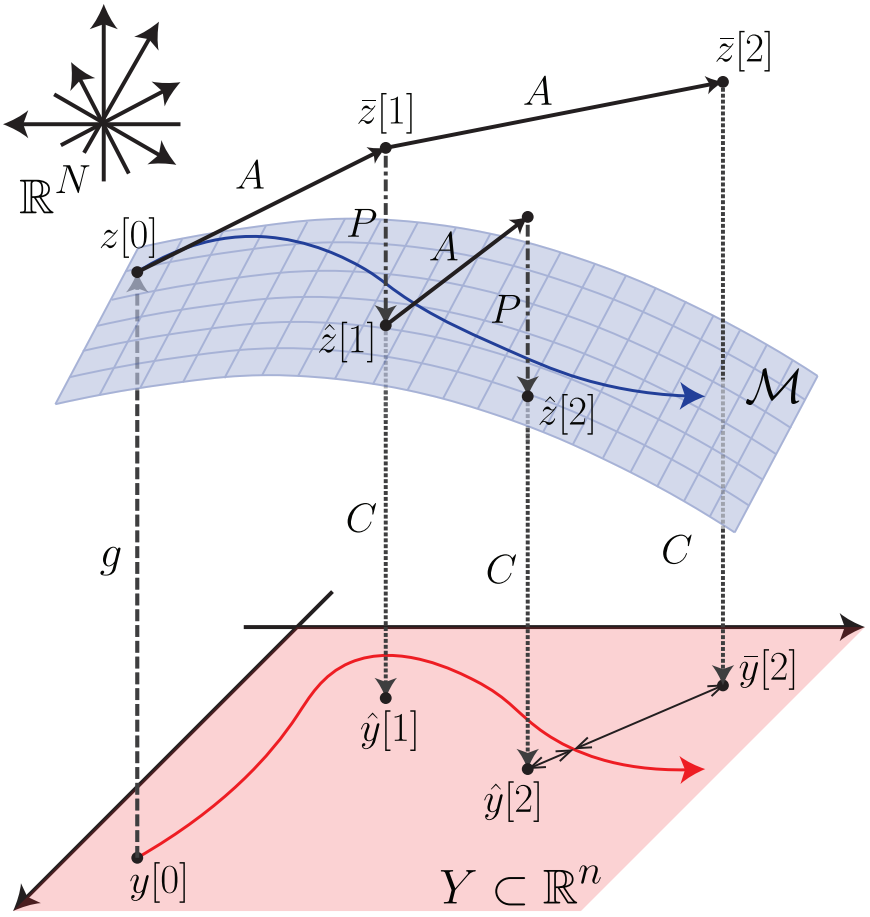


Figure 3.2: An illustration of the effect of deviating from the image of the lifting function $\mathcal{M}$ and how it can be remedied by defining a projection operation as described in Section 3.1.4. The evolution of the finite-dimensional system in the output space Y from $\boldsymbol{y}[0]$ is depicted 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 $\boldsymbol{z}[i]$ may generate a solution that is outside of $\mathcal{M}$. Though one can still apply C to $\bar{\boldsymbol{z}}$ to project it back to Y, this may result in poor performance. Instead, by projecting $\bar{\boldsymbol{z}}[i]$ onto the manifold at each discrete time step to define a new lifted state $\hat{\boldsymbol{z}}[i]$, the deviation from $\mathcal{M}$ is reduced, which improves overall predictive performance.

overview of the LASSO method and its implementation see [69].

The benefit of using L^1-regularization to reduce overfitting rather than L^2-regularization (e.g. ridge regression) comes from 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). In addition to reducing model complexity, sparsity is also desirable because it reduces the memory needed to store model matrices on a computer, enabling a higher dimensional set of basis functions to be used to construct the lifting function $\boldsymbol{\psi}$.

Though sparsity is desirable, it can carry a cost in prediction accuracy. Consider a linear realization of the form (3.19) and let $\mathcal{M}$ denote the image of $\boldsymbol{g}$, which maps an output $\tilde{\boldsymbol{y}}$ to its corresponding lifted state $\tilde{\boldsymbol{z}}$. Because A, B are data-driven approximations, there is no guarantee that $A\boldsymbol{g}(\boldsymbol{y}[i]) + B\boldsymbol{u}[i]$ will map onto $\mathcal{M}$ for all i. In other words, advancing the lifted state forward in time using the identified linear model matrices A, B may cause it to leave the space of "legitimate" lifted states. Deviating from this space will incur prediction error when projecting back down into the output space. This effect is illustrated in Fig. 3.2 for a system with zero input. As model sparsity increases, so does the magnitude of deviations from $\mathcal{M}$, and consequently prediction error increases as well. To minimize this prediction error induced by sparsity, we desire a way to minimize the distance from $\mathcal{M}$ at each iteration.

For a linear model realization in the form of (3.19), 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

$$P\left(A\boldsymbol{g}(\boldsymbol{a}^{(k)}) + B\boldsymbol{u}^{(k)}\right) = \boldsymbol{g}(\boldsymbol{b}^{(k)}) \tag{3.49}$$

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

$$\Omega_a := \begin{bmatrix} \left(A\boldsymbol{g}(\boldsymbol{a}^{(1)}) + B\boldsymbol{u}^{(1)}\right)^\top \\ \vdots \\ \left(A\boldsymbol{g}(\boldsymbol{a}^{(K)}) + B\boldsymbol{u}^{(K)}\right)^\top \end{bmatrix}. \tag{3.50}$$

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

$$P := \left(\Omega_a^\dagger \Psi_b\right)^\top. \tag{3.51}$$

Composing P with the A and B matrices in (3.19) yields a modified linear model that reduces the distance from $\mathcal{M}$ at each iteration,

$$z[i+1] = \hat{A}z[i] + \hat{B}u[i] \tag{3.52}$$

where $\hat{A} := PA$ and $\hat{B} := PB$. Algorithm 1 summarizes the proposed linear model construction process with this added projection.

For a bilinear model realization in the form of (3.28), the same procedure can be applied. In this case, the ideal projection operator P should satisfy the following for all k

$$P\left(A\boldsymbol{g}(\boldsymbol{a}^{(k)}) + B\boldsymbol{u}^{(k)} + \sum_{j=1}^{m} H_j \boldsymbol{g}(\boldsymbol{a}^{(k)}) u_j^{(k)}\right) = \boldsymbol{g}(\boldsymbol{b}^{(k)}). \tag{3.53}$$

If we construct a $K \times N$ matrix

$$\Omega_a := \begin{bmatrix} \left(A\boldsymbol{g}(\boldsymbol{a}^{(1)}) + B\boldsymbol{u}^{(1)} + \sum_{j=1}^{m} H_j \boldsymbol{g}(\boldsymbol{a}^{(1)}) u_j^{(1)}\right)^\top \\ \vdots \\ \left(A\boldsymbol{g}(\boldsymbol{a}^{(K)}) + B\boldsymbol{u}^{(K)} + \sum_{j=1}^{m} H_j \boldsymbol{g}(\boldsymbol{a}^{(K)}) u_j^{(K)}\right)^\top \end{bmatrix}, \tag{3.54}$$

then the best projection operator in the L^2-norm sense based on our data is given by (3.51). Com-

posing P with the A, B, and $\{H_j\}_{j=1}^m$ matrices in (3.28) yields a modified bilinear model that reduces the distance from $\mathcal{M}$ at each itearion,

$$z[i+1] = \hat{A}z[i] + \hat{B}u[i] + \sum_{j=1}^{m} \hat{H}_j z[i]u_j[i] \tag{3.55}$$

where where $\hat{A} := PA$ and $\hat{B} := PB$, and $\hat{H}_j := PH_j$. Algorithm 2 summarizes the proposed bilinear model construction process with this added projection.

The projection procedure is not necessary for nonlinear model realizations because they advance the value of the output directly rather than indirectly via a lifted state. The output is not lifted to a higher dimensional space, thus there is no danger of future predictions accumulating error by leaving the set of legitimate lifted states.

3.1.4.2 Dimensional Reduction

The effectiveness of a finite-dimensional approximation $\bar{\mathcal{K}}_{T_s}$ in representing the actions of the infinite-dimensional Koopman operator $\mathcal{K}_{T_s}$ depends greatly upon the the span of the basis functions selected (i.e. the subspace $\bar{\mathcal{F}}$). We can use data to intelligently inform our choice of basis functions, rather than choosing them naively. This enables us to reduce the dimension of Koopman models without significantly impacting accuracy, thus making them more computationally tractable.

This process consists of computing the singular value decomposition (SVD) of a data matrix of lifted experimental measurements to find an orthogonal basis for the finite-dimensional function space $\bar{\mathcal{F}}$. A new lower-dimensional subspace $\tilde{\mathcal{F}}$ spanned by the basis functions associated with the largest singular values can then be used for Koopman-based system identification.

The first step in this dimensional reduction process is to lift the snapshots $\{\boldsymbol{a}^{(k)}\}_{k=1}^K$ using an

initial set of N basis functions $\{g_i\}_{i=1}^N$ and compile them into a $K \times N$ data matrix, denoted by Υ,

$$\Upsilon = \begin{bmatrix} g_1(a^{(1)}) & \cdots & g_N(a^{(1)}) \\ \vdots & & \vdots \\ g_1(a^{(K)}) & \cdots & g_N(a^{(K)}) \end{bmatrix}. \tag{3.56}$$

If we then compute the SVD of this matrix,

$$\Upsilon = U_\Upsilon \Sigma_\Upsilon V_\Upsilon^\top, \tag{3.57}$$

the columns of V_Υ, also called the right singular vectors, provide an orthonormal basis for the row space of Υ, and the corresponding singular values provide a measure of the correlation between the data and each of the vectors. Each right singular vector $\boldsymbol{v}_i = \begin{bmatrix} v_1 & \cdots & v_N \end{bmatrix}^\top$ also describes a function h_i as a linear combination of the initial basis functions $\{g_i\}_{i=1}^N$,

$$h_i = v_1 g_1 + \cdots + v_N g_N \tag{3.58}$$

We could define a new basis for the function space $\bar{\mathcal{F}}$ as the set of all such functions $\{h_i\}_{i=1}^N$, and use this basis to identify an approximation of the Koopman operator. However, doing so would have no computational benefit since the dimension is not reduced.

To achieve a dimensional reduction, we use the so-called "truncated SVD" instead. The truncated SVD is similar to the full SVD, but it yields a decomposition of a lower rank approximation of Υ rather than Υ itself. This is achieved by only retaining the subset of left and right singular vectors corresponding to singular values above some threshold [70]. This reduced number $N' < N$ of right singular vectors can then be used to construct a basis for an N' dimensional function space $\tilde{\mathcal{F}}$, and an approximation of the Koopman operator in that space can be identified via the methods presented in the preceding sections.

3.2 Application: Planar 3-link Arm

We claim that the Koopman operator can be used to construct accurate models of soft robotic systems from data. However, before applying the system identification methods described in Section 3.1 to a soft robotic system, we applied them to the simulated planar arm system shown in Fig. 3.3. This simpler example is used to explore and highlight the differences between linear, bilinear, and nonlinear model realizations of the same system.

The arm has 3-links each of mass 100g and length 0.33m and 3 joints each with a stiffness of 1×10^{-5} N/rad and a viscous damping coefficient of 1 Ns/rad. The input into the system is a set of $m = 3$ applied joint torques and the output is the location of the end of each link expressed as a $n = 6$ dimensional vector of Cartesian coordinates,

$$\boldsymbol{u}(t) = \begin{bmatrix} \tau_1(t) & \tau_2(t) & \tau_3(t) \end{bmatrix}^\top \tag{3.59}$$

$$\boldsymbol{y}(t) = \begin{bmatrix} \alpha_1(t) & \beta_1(t) & \alpha_2(t) & \beta_2(t) & \alpha_3(t) & \beta_3(t) \end{bmatrix}^\top \tag{3.60}$$

as shown in Fig. 3.3. A set of 12000 snapshots was collected for a time step of $T_s = 0.05$ seconds over a range of randomized initial conditions and inputs for the purposes of identifying matrix approximations of the Koopman operator as well as linear, bilinear, and nonlinear model realizations of the system.

3.2.1 Model Prediction Comparison

The predictive accuracy of various models identified using the Koopman approach depends on both the model type (e.g. linear, bilinear, or nonlinear) as well as the number of basis functions chosen. We identified several models according to Algorithms 1, 2, and 3 using monomial basis functions.

The sets of basis functions used for identifying the linear models consisted of all monomials of the elements of the output up to a specific degree denoted ρ, plus the indicator functions on each

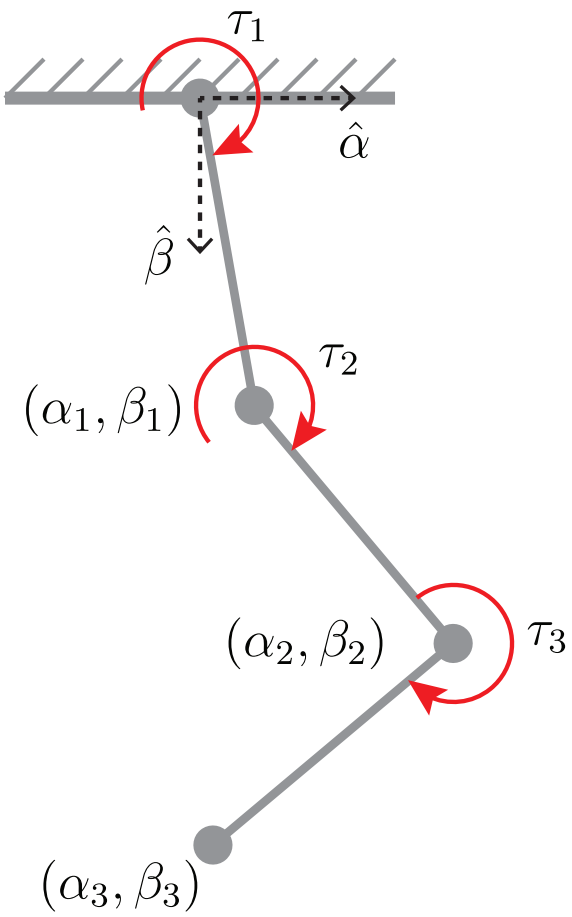


Figure 3.3: Three link planar arm system with input defined as joint torques and output defined as the locations of the end of each link.

component of the input, i.e.

$$\{\psi_i(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{u}})\}_{i=1}^{N} = \left\{\tilde{y}_1^{\rho_1}\cdots\tilde{y}_n^{\rho_n}\middle|(\rho_1,...,\rho_n)\in\mathbb{N}_0^n,\rho_1+\cdots+\rho_n\leq\rho\right\}\cup\left\{\tilde{u}_i\middle|i\in\{1,...,m\}\right\} \tag{3.61}$$

where $N=(n+\rho)!/(n!\rho!)+m$, and $\mathbb{N}_0$ indicates the set of natural numbers including zero. The sets of basis functions used for identifying bilinear models consisted of the same monomials as well as the product of each monomial with each component of the input, i.e.

$$\{\psi_i(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{u}})\}_{i=1}^{N(m+1)} = \left\{(\tilde{y}_1^{\rho_1}\cdots\tilde{y}_n^{\rho_n})\hat{u}\middle|(\rho_1,...,\rho_n)\in\mathbb{N}_0^n,\rho_1+\cdots+\rho_n\leq\rho,\hat{u}\in\{1,\tilde{u}_1,...,\tilde{u}_m\}\right\} \tag{3.62}$$

where $N=(n+\rho)!/(n!\rho!)$. The sets of basis functions used for identifying discrete nonlinear models consisted of monomials up to degree ρ of both the input and the output, i.e.

$$\{\psi_i(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{u}})\}_{i=1}^{N} = \left\{(\tilde{y}_1^{\rho_1}\cdots\tilde{y}_n^{\rho_n})(\tilde{u}_1^{\rho_{n+1}}\cdots\tilde{u}_m^{\rho_{n+m}})\middle|(\rho_1,...,\rho_{n+m})\in\mathbb{N}_0^{(n+m)},\rho_1+\cdots+\rho_{n+m}\leq\rho\right\} \tag{3.63}$$

where $N=(n+m+\rho)!/((n+m)!\rho!)$.

Six linear models were identified for values of $\rho=1,..,6$. Four bilinear and four nonlinear models were identified for values of $\rho=1,...,4$. Table 3.1 indicates the total number of basis functions, i.e. $\dim\big(\boldsymbol{\psi}(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{u}})\big)$, used for identifying the Koopman operator matrix for each model. The prediction accuracy of each model was evaluated by comparing model simulations to validation data and computing the average error over all validation data points. This error was quantified as the Euclidean distance between the predicted and real outputs in $\mathbb{R}^6$.

Fig. 3.4 displays the prediction error verses the dimension of the Koopman operator matrix for each model. The error shown in the plot is normalized by the average error incurred by the zero response. The accuracy of the linear model increases very little, even when the dimension of the system is increased by several orders of magnitude. The bilinear model become more accurate as

Table 3.1: Number of Monomial Basis Functions for Identified Models

ρ	# **of basis functions, i.e. dim**$\big(\psi(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}})\big)$		
	Linear	Bilinear	Nonlinear
1	10	28	10
2	31	112	55
3	87	336	220
4	213	-	715
5	465	-	-
6	927	-	-

the dimension increases, but eventually becomes unstable. The nonlinear model becomes more accurate as the dimension increases and does not become unstable.

3.2.2 Discussion

The noticeable differences in performance between the linear, blinear, and nonlinear models stems from the differences in the sets of basis functions used in the system identification process. Because the Koopman operator is linear, Koopman-based system identification methods yield a discrete flow map that is a linear combination of basis functions. Hence, if the true discrete flow map of the system contains nonlinear terms that are not included in the set of basis functions, the identified system model will be limited in its ability to capture the true system behavior.

Based on the results shown in Fig. 3.4, the nonlinear Koopman modeling approach appears to be the best choice. Indeed, a nonlinear model achieves the lowest prediction error overall, and the nonlinear models do not become unstable for the larger sets of basis functions. Their accuracy can be attributed to the richness of the sets of basis functions used for identification, and their stability is perhaps due to the fact that the nonlinear systems flow the output forward in $\mathbb{R}^6$ rather than flowing a lifted state forward in $\mathbb{R}^N$ where there is more opportunity to overfit.

The downside of nonlinear models is that they are not compatible with efficient linear control techniques. Therefore, the nonlinear models generated using the Koopman operator may be the best option when open-loop prediction accuracy is desired, but will be of little value in synthesizing computationally efficient closed-loop controllers.

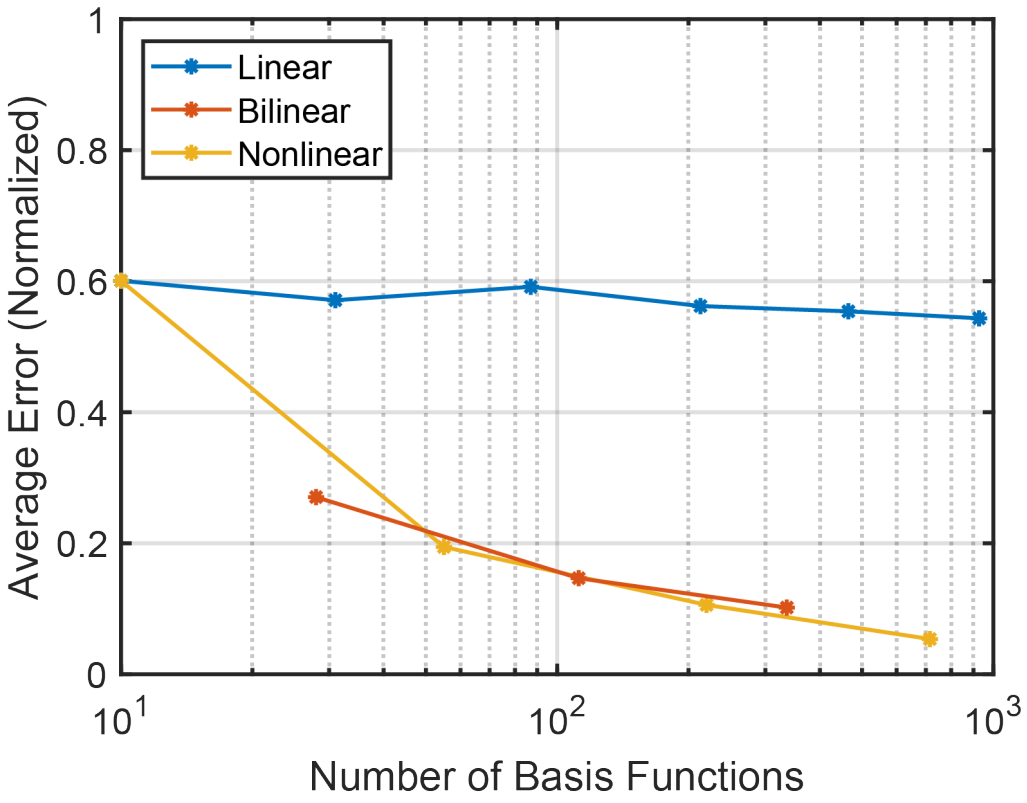


Figure 3.4: The model prediction error for several linear, bilinear, and nonlinear Koopman model realizations identified from the same set of data. As the number of basis functions increases, the error of the linear model changes little, the error of the bilinear model decreases before becoming unstable, and the the error of the nonlinear model decreases monotonically.

Linear models are compatible with a myriad of efficient and well-established closed-loop control design techniques. For this system, however, the linear models perform poorly compared to the bilinear and nonlinear models. Furthermore, prediction accuracy does not significantly improve with increases to the number of basis functions. This seems to indicate that linear combinations of the type of basis functions described by (3.61) may be fundamentally incapable of approximating some non-linear terms of the true system dynamics.

The bilinear models exhibit accuracy nearly identical to that of the nonlinear models and become more accurate as the dimension increases, up to a point. Eventually one of the bilinear models becomes unstable and the prediction error increases dramatically. This can likely be attributed to over-fitting. While the basis functions described by (3.62) are capable of approximating nonlinearities that the linear model basis functions fail to capture, including too many of them can lead to an overly complex model. Such over-fitting could be mitigated by applying the techniques presented in Section 3.1.4.

The bilinear models exhibit predictive performance on par with the nonlinear models, while being significantly less complex. Because the bilinear models use basis functions that do not include any higher order input terms, they reduce to linear models for a fixed value of the output. This feature can be exploited to synthesize computationally efficient linear controllers from bilinear models, simultaneously taking advantage of the accuracy of bilinear models while retaining the computational tractability of linear control algorithms. This topic of utilizing bilinear system models for control applications will be more fully explored in Section 4.1.3.

3.3 Application: Pneumatic Soft Robot Arm

To evaluate how well the Koopman-based modeling approach stacks up against the current state of the art on a real soft robotic system, we applied the Koopman nonlinear system identification method (Algorithm 4) to a continuously deformable soft robot arm and compared the resulting model to those generated by several other nonlinear system identification techniques. In the fol-

lowing, we describe in detail the robotic hardware, experimental setup, measurement procedure, and the data processing involved in the system identification process, as well as the process by which performance was evaluated and compared across models.

3.3.1 Hardware Description

The soft robot used in this experiment consisted of three pneumatically driven McKibben actuators (also known as Pneumatic Artificial Muscles or PAMs) adhered together by latex rubber and connected to a common base mount on one end and to an end effector on the other (see Fig. 3.5). During the trials, the pressure inside each actuator was varied using a pneumatic pressure regulator (Enfield TR-010-g10-s), and the displacement and velocity of the end effector was measured at a frequency of 60 Hz using a commercial motion capture system (Phase Space Impulse X2E).

For this robot, it is our primary interest to control the motion of the end effector. Hence, we chose an output which is capable of describing the dynamics of the end effector as an ordinary differential equation, namely the position and velocity of the end effector with respect to a global coordinate frame, as shown in Fig. 3.5

$$\boldsymbol{y}(t) = \begin{bmatrix} y_1(t) & y_2(t) & y_3(t) & \dot{y}_1(t) & \dot{y}_2(t) & \dot{y}_3(t) \end{bmatrix}^\top \tag{3.64}$$

3.3.2 Data Collection

To generate a representative sampling of the system's behavior over its entire operation range, a randomized input was applied. The control input into the system $\boldsymbol{u}$ was a set of three $0-10$V signals into the pressure regulators corresponding to actuator pressures of $\approx 0-140$ kPa

$$\boldsymbol{u}(t) = \begin{bmatrix} u_1(t) & u_2(t) & u_3(t) \end{bmatrix}^T, \qquad u_i \in [0, 10] \tag{3.65}$$

Before each trial, a $3 \times K_u$ table Υ of uniformly distributed random numbers between zero and ten was generated to be used as an input lookup table. K_u was chosen to be large enough to

Table 3.2: Data Collection Parameters

	Trial					
	1	2	3	4	5	6
Length (min)	40:06	31:26	8:16	26:09	28:38	31:35
$\boldsymbol{T_u}$ **(s)**	3.0	3.5	4	5	5	8

provide inputs over the entire length of each trial. Each control input was smoothly 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,

$$u_i(t) = \frac{(\Upsilon_{i,k+1} - \Upsilon_{i,k})}{T_u}\left(t + \frac{(i-1)T_u}{3}\right) + \Upsilon_{i,k} \tag{3.66}$$

where $k = \text{floor}\left(t/T_u\right)$ is the current index into the lookup table at time t.

Data collection proceeded in 6 trials each lasting an average of $\approx$ 27 minutes. The input transition period T_u varied from trial to trial, taking values between 3 and 8 seconds (see Table 3.2 for the specific values for each trial). After collecting data, raw position and velocity measurements were put through a moving average filter with window size of 1s to reduce noise, then sampled uniformly with a period $T_s = 0.02$ seconds. Sampled velocity measurements were put through a second moving average filter with window size of 1 second due to the higher noise content of the velocity signal. The time-series data from each trial was partitioned into training and validation sets. Three 10 second validation sets were extracted from each trial and the remainder of the trial data was used for training.

3.3.3 Model Comparison

We generated a state space model from the technique described by Algorithm 4 using a monomial basis of maximum degree $\rho = 3$ and the collected training data. We then evaluated its accuracy by comparing model simulations to each of the validation data sets (Fig. 3.6). Goodness of fit for the trajectory of the j^{th} component of the output y_j was calculated using the normalized root-mean-

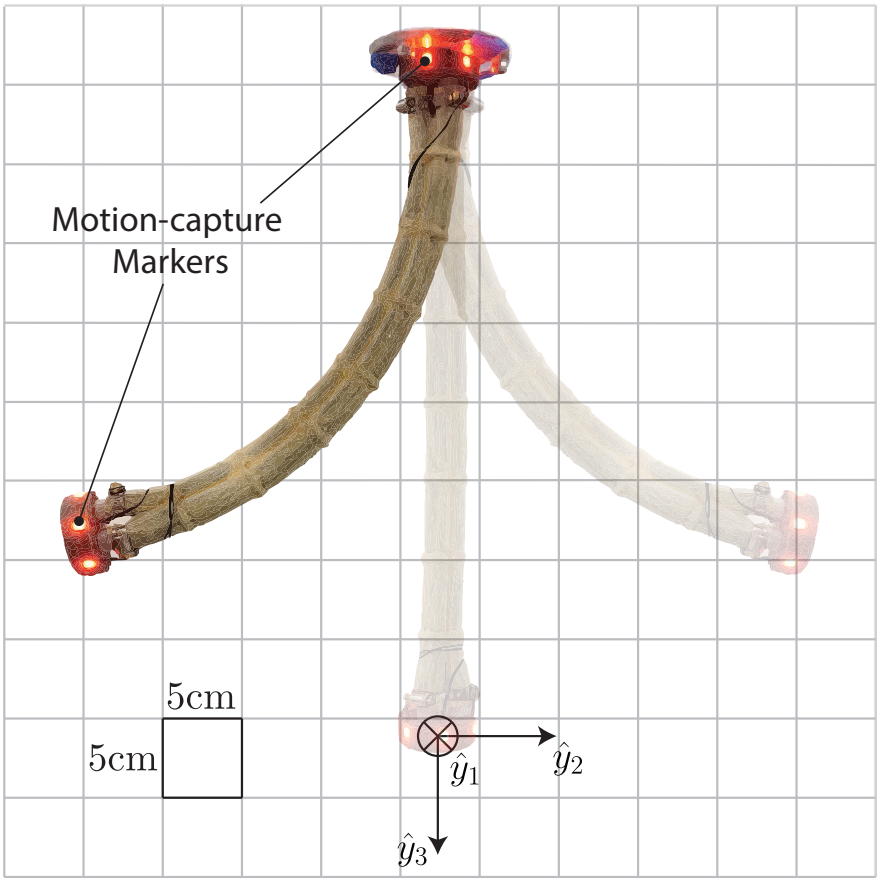


Figure 3.5: System identification was performed on a soft robot consisting of three PAMs adhered together. Active motion capture markers on the base and end effector enabled tracking of the position and velocity of the end effector relative to the fixed global coordinate frame marked by unit vectors $\hat{y}_1, \hat{y}_2, \hat{y}_3$ where $\hat{y}_1$ is pointing into the page. The robot's range of motion given control inputs defined in (4.14) is depicted.

square error (NRMSE), defined:

$$\text{RMSE} = \sqrt{\frac{\sum_{k=1}^{N_{\text{total}}} \left(y_j[k] - \hat{y}_j[k]\right)^2}{N_{\text{total}}}} \tag{3.67}$$

$$\text{NRMSE} = \left(\frac{\text{RMSE}}{y_{j,\text{max}} - y_{j,\text{min}}}\right) \cdot 100\% \tag{3.68}$$

where $\hat{y}_j$ is the simulated value of the j^{th} component of output, N_{total} is the total number of points, and $y_{j,\text{min/max}}$ are the measured minimum/maximum values of the j^{th} component of the output observed over all trials.

The performance of our model was benchmarked against a linear state space, nonlinear Hammerstein Wiener, nonlinear auto-regressive with exogenous inputs (NLARX), and a feedforward neural network model. The models were trained and evaluated on the same non-lifted time-series data as the Koopman model, and generated using either the Matlab System Identification Toolbox or Neural Network Toolbox [71]. The state space model was generated using the subspace method [62, Chapter 7] and specified to be 6 dimensional, i.e. the same dimension as the state defined in 3.64. The neural network model was trained using the Levenberg-Marquardt backpropogation algorithm and sigmoid activation functions. It was trained several times using combinations of 10-30 hidden neurons and 1-10 delays. Only the results for the best of these models, corresponding to 10 hidden neurons and 10 delays, is displayed in Fig. 3.7 and Table 4.2.

3.3.4 Results

The model generated by the Koopman system identification method has a total RMSE averaged across position states and velocity states of 5.98 mm and 3.66 mm/s, respectively. As shown in Table 4.2, this corresponds to a total NRMSE averaged across all states of 2.1%. By this metric, it performs more than twice as well as the best competing linear and nonlinear models which have average NRMSEs of 4.6% and 4.5%, respectively (see Fig. 3.7). The Koopman model also exhibits the smallest standard deviation of the NRMSE across states. This implies that the Koopman model

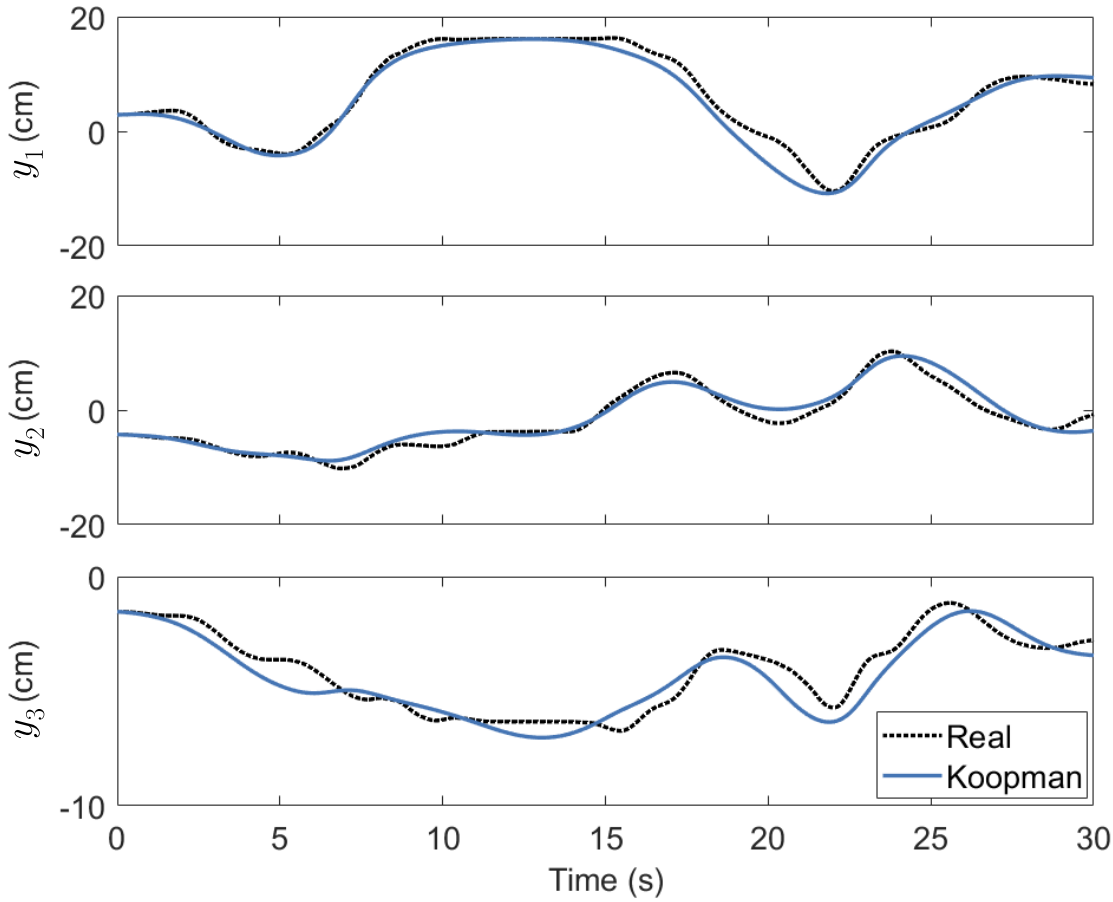


Figure 3.6: The measured position of the robot end effector over a 30 second time window (black, dotted) superimposed with the position predicted by the Koopman-based model (blue) given the same initial condition and control inputs. Coordinates are defined with respect to the global coordinate frame depicted in Fig. 3.5.

more consistently captures the real behavior of all six states of the system, rather than just a subset of them. Fig. 3.6 illustrates the ability of the Koopman-based model to predict the position of the end effector over a 30 second time horizon.

3.3.5 Discussion

We have successfully applied a system identification technique based on Koopman operator theory to a soft robot and shown that the model generated outperforms those constructed by several other state-of-the-art nonlinear system identification methods. Perhaps unsurprisingly, the linear state space model was unable to capture the nonlinear dynamics of the robot as well as the Koopman

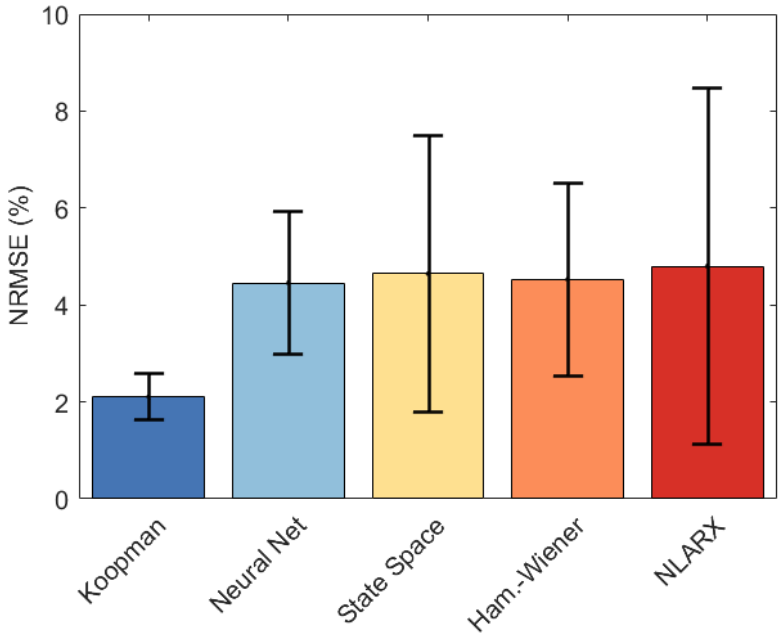


Figure 3.7: Shown is the total NRMSE averaged across all states for each of the models, with the standard deviation designated by the black bar. The average NRMSE of the Koopman-based model is less than half of that of the other models, with a standard deviation of less than one third of the other models.

Table 3.3: Total NRMSE (%) over all validation trials

Model	**States**						**Avg.**	**Std. Dev.**
	y_1	y_2	y_3	y_4	y_5	y_6		
Koopman	2.4	2.0	2.9	1.7	1.5	2.0	2.1	0.5
Neural Net	5.8	4.0	6.6	3.9	2.8	3.5	4.5	1.5
State Space	5.1	3.1	9.9	3.0	1.8	4.8	4.6	2.9
Ham.-Weiner	7.0	4.5	6.9	3.0	2.3	3.1	4.5	2.0
NLARX	5.0	3.0	12.0	3.8	2.1	2.8	4.8	3.7

model. As for the nonlinear models, there are several likely reasons why the performance of the Koopman model was superior. Since the Koopman model is a state space model, simulations can be initialized from the same initial condition as the real system. This is not the case for the other learned nonlinear models which do not have an internal state corresponding to the physical state of the robot. Rather, they act as black-box models only capable of mapping inputs to outputs.

Another advantage of the Koopman model is that its quality does not depend on an initial model estimate or tuning parameters. By iterating over the set of all initializations and tuning parameters, one may be able to generate better performing models than those shown; unfortunately, this multivariate trial-and-error process may not affect results in a predictable way. In contrast, the only tuning parameter involved in the Koopman method is the maximum degree of the monomial basis functions, which has a magnitude that is directly proportional to model accuracy.

While the results here are promising, there are practical challenges to extending the Koopman approach to higher dimensional systems. As the dimension of the state space increases, so does the size of the monomial basis set of the finite-dimensional subspace of observables. This greatly increases the size of the matrix equations that must be solved, leading to computational intractability for sufficiently high dimensional systems. However, if some information about the system is known beforehand, this issue could be counteracted by choosing a more suitable basis for the observables. For example, if the system exhibits oscillatory motion, a lower dimensional fourier basis may be more suitable than a monomial basis to represent the behavior. Such an extension of the method is left to future work.

CHAPTER 4

Model-based Control

Soft robots differ from rigid-bodied robots in several ways that make them uniquely difficult to control. Namely, soft robots exhibit distributed rather than localized deformation and actuation, soft materials exhibit nonlinear characteristics that are negligible in rigid materials, and soft robots span such a wide range of designs that it is difficult to generalize control results from one robot to another. These differences render many of the standard approaches used to control rigid-bodied robots insufficient for soft robots. Thus, novel control approaches for soft robots are needed.

Model-free control approaches have emerged as a convenient way to construct control policies directly from data, without first identifying a system model. For instance, [27] used deep reinforcement learning to achieve open-loop position control of a soft manipulator comprised of fiber-reinforced actuators, and [29] used a combination of a recurrent neural network and supervised reinforcement learning to achieve closed-loop control of a pneumatically-driven soft manipulator. These data-driven approaches rely on machine learning techniques to identify suitable control policies for a specific task, but without an underlying system model, they do not generalize well to arbitrary tasks.

A system model enables the design of model-based controllers that leverage model predictions to choose suitable control inputs for arbitrary tasks. In particular, model-based controllers can anticipate future events, allowing them to optimally choose control inputs and facilitate better control performance. When an accurate model is available, predictive controllers can be built by using the model to calculate a feedforward term, then adding a feedback term to account for

minor model uncertainty and disturbances. If an accurate model is unavailable, feedback must be relied upon more heavily. This poses several problems for soft robots. First, feedback requires sensing, but the morphology of soft robots precludes the use of most conventional sensors. Suitable alternatives are currently in development [72, 73, 74, 75], but are not yet readily available. Second, relying heavily on feedback to compensate for an inaccurate model has been illustrated to reduce the compliance of soft robotic systems [76]. That is, excessive feedback negates the desirable compliance of a soft robot by replacing its natural dynamics with those of a slower, stiffer system. Therefore, accurate models are required to control soft robots in a manner that reduces dependence on feedback and simultaneously preserves compliance.

Many model-based controllers have been developed for soft robots, but the majority of them are static and rely on simplifying assumptions such as piece-wise constant curvature [18]. Such controllers have proven sufficient for static control of uniform, low-mass manipulators, but they have been insufficient for dynamic control of more complex soft robotic systems [21]. Dynamic control of a soft manipulator has been achieved by supplementing a piece-wise constant curvature model with data-driven trajectory optimization [77], but the approach requires task specific training. Few model-based controllers have been derived from more realistic physics-based models due to their computational complexity.

As seen in Chapter 3, Koopman operator theory offers a way to construct linear or bilinear model realizations of nonlinear dynamical systems from data. Such realizations are computationally efficient without relying on physical simplifying assumptions. Several researchers have exploited these properties to construct model-based controllers from Koopman representations [31]. In [34] they construct an LQR feedback law from a linear Koopman representation and use it to successfully control the the underlying nonlinear dynamical system, and a similar approach is utilized in [38] to control a swimming fish robot. In [36], linear Koopman realizations are combined with model predictive control (MPC), a popular model-based control design technique wherein one optimizes the control input over a finite time horizon, applies that input for a single time-step, and then optimizes again, repeatedly [78].

This chapter builds upon those contributions within the specific context of soft robot control. We develop several Koopman-based model predictive control algorithms and demonstrate their efficacy in controlling a real soft robotic system. In Section 4.1 we formally introduce model predictive control and describe methods for constructing MPC controllers from Koopman model realizations. In Section 4.2, we construct a linear, bilinear, and nonlinear MPC controller for the simulated planar 3-link arm system from Section 3.2, and compare the performances of the controllers in a trajectory following task. In Section 4.3, we construct several MPC controllers for a real soft robot arm, and evaluate the efficacy of each controller in performing trajectory following tasks. Our results show that Koopman-based MPC controllers outperform benchmark controllers for soft robotic systems. Most of the contents of this chapter originally appeared in [41], but an attempt has been made to clarify and expand upon the original exposition of the material. Notably, this work introduces nonlinear and suboptimal bilinear MPC in addition to linear MPC and presents new simulation results which were absent in the original text.

4.1 Model Predictive Control

Model predictive control algorithms optimally choose a sequence of control inputs given a desired output trajectory and system model. This system model can be either linear or nonlinear, but linear models have computational advantages over nonlinear ones. Namely, the MPC optimization problem for linear models is convex, while for nonlinear models it is not.

Convex optimization problems, i.e. those which have a convex cost function and convex feasible set, have global extrema and can be solved very reliably and efficiently using interior-point methods or other special methods for convex optimization [30]. Non-convex optimization problems, on the other hand, may have multiple local extrema and there is no standard method for finding global solutions. The most efficient non-convex optimization methods only find local solutions, which depend on an initial guess for the optimization variable. Thus, for real-time optimal control problems, convexity is desired.

The MPC optimization problem for linear systems is convex because it has a quadratic cost function and linear constraints. Since it is convex, it has a unique globally optimal solution that can efficiently be constructed without initialization even for models with thousands of states and inputs [30, 79, 80]. This contrasts sharply with the MPC formulation for nonlinear systems which requires solving an optimization problem with nonlinear constraints and a (potentially) nonlinear cost function [81]. As a result, algorithms to solve such problems typically require initialization and can struggle to find globally optimal solutions [82]. Though techniques have been proposed to improve the speed of algorithms to solve nonlinear MPC problems [83, 84] or even globally solve such problems without requiring initialization [85], these formulations still take orders of magnitude more time per iteration, which can make them too slow to be applied for real-time control. This difference is highlighted in Sections 4.2 and 4.3 via computation time comparisons between linear and nonlinear MPC controllers for several robotic systems.

The remainder of this section describes the formulation of MPC algorithms for linear and nonlinear Koopman realizations, as well as a suboptimal MPC control algorithm for bilinear Koopman realizations. These algorithms, when combined with Koopman-based models derived using the methods of Chapter 3, can be used to control real soft robotic systems.

4.1.1 Linear MPC

For linear systems, MPC consists of iteratively solving a convex quadratic program. This is also the case for Koopman-based model predictive control (K-MPC), wherein one solves for the optimal sequence of control inputs over a receding horizon according to the following quadratic program

Algorithm 5: Koopman-based Linear MPC (K-MPC)

Input: Prediction horizon: N_h
Cost matrices: $Q[i], R[i], q[i], r[i]$ for $i = 0, ..., N_h$
Constraint matrices: $E[i], F[i], b_i$ for $i = 0, ..., N_h$
Model matrices: A, B

for $k = 0, 1, 2, ...$ **do**
- **Step 1:** Set $\acute{\boldsymbol{z}}[0] = g(y[k])$
- **Step 2:** Solve (4.1) 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

end

at each time instance k of the closed-loop operation:

$$\begin{aligned}
\min_{\substack{\{\acute{\boldsymbol{u}}[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{\acute{\boldsymbol{z}}[i]\in\mathbb{R}^N\}_{i=0}^{N_h}}} \quad & \acute{\boldsymbol{z}}[N_h]^T Q[N_h]\acute{\boldsymbol{z}}[N_h] + q[N_h]^T\acute{\boldsymbol{z}}[N_h] + \\
& + \sum_{i=0}^{N_h-1}\Big(\acute{\boldsymbol{z}}[i]^T Q[i]\acute{\boldsymbol{z}}[i] + \acute{\boldsymbol{u}}[i]^T R[i]\acute{\boldsymbol{u}}[i] + q[i]^T\acute{\boldsymbol{z}}[i] + r[i]^T\acute{\boldsymbol{u}}[i]\Big) \\
\text{s.t.} \quad & \acute{\boldsymbol{z}}[i+1] = A\acute{\boldsymbol{z}}[i] + B\acute{\boldsymbol{u}}[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& E[i]\acute{\boldsymbol{z}}[i] + F[i]\acute{\boldsymbol{u}}[i] \leq b[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& \acute{\boldsymbol{z}}[0] = \boldsymbol{g}(\boldsymbol{y}[k])
\end{aligned} \tag{4.1}$$

where $N_h \in \mathbb{N}$ is the prediction horizon, $Q[i] \in \mathbb{R}^{N\times N}$ and $R[i] \in \mathbb{R}^{m\times m}$ 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 (4.1) depend on the dimension of the lifted state N, [36] shows that these can be rendered independent of N by transforming the problem into its so-called "dense-form." Algorithm 7 summarizes the closed-loop operation of this Koopman-based linear MPC controller.

4.1.2 Nonlinear MPC

For nonlinear systems, nonlinear model predictive control (NMPC) typically consists of iteratively solving a non-convex optimization problem. Non-convexity is introduced by the set of nonlinear constraints that prohibit the decision variables from violating the system's dynamics (highlighted in red in (4.2) and (4.6)). NMPC algorithms exist for both discrete and continuous nonlinear dynamical systems. Their general formulations are presented in the following subsections.

4.1.2.1 Discrete NMPC

For discrete nonlinear systems, the NMPC cost function and state and input polyhedral constraints are identical in form to those of linear MPC, but the constraints that enforce the dynamics are not linear. For Koopman-based nonlinear model predictive control (K-NMPC) one solves for the optimal sequence of control inputs over a horizon according to the the following nonlinear program starting from the k^{th} time-step of the system:

$$\begin{aligned}
\min_{\substack{\{\acute{\boldsymbol{u}}[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{\acute{\boldsymbol{y}}[i]\in\mathbb{R}^n\}_{i=0}^{N_h}}} \quad & \acute{\boldsymbol{y}}[N_h]^T Q[N_h]\acute{\boldsymbol{y}}[N_h] + q[N_h]^T\acute{\boldsymbol{y}}[N_h] + \\
& + \sum_{i=0}^{N_h-1}\left(\acute{\boldsymbol{y}}[i]^T Q[i]\acute{\boldsymbol{y}}[i] + \acute{\boldsymbol{u}}[i]^T R[i]\acute{\boldsymbol{u}}[i] + q[i]^T\acute{\boldsymbol{y}}[i] + r[i]^T\acute{\boldsymbol{u}}[i]\right) \\
\text{s.t.} \quad & \acute{\boldsymbol{y}}[i+1] = T(\acute{\boldsymbol{y}}[i], \acute{\boldsymbol{u}}[i]), \quad \forall\{i\}_{i=0}^{N_h-1} \\
& E[i]\acute{\boldsymbol{y}}[i] + F[i]\acute{\boldsymbol{u}}[i] \leq b[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& \acute{\boldsymbol{y}}[0] = \boldsymbol{y}[k]
\end{aligned} \tag{4.2}$$

where $N_h \in \mathbb{N}$ is the prediction horizon, $Q[i] \in \mathbb{R}^{n\times n}$ and $R[i] \in \mathbb{R}^{m\times m}$ are positive semidefinite matrices, and where each time the program is called, the predictions are initialized from the current output $y[k]$.

The time required to solve this problem depends on the dimension of the model, the length of the horizon, the complexity of the dynamics, and the efficiency of the optimization software used. Often the solving time will exceed the time-step of the the discrete dynamical model, mak-

ing it impossible to update the control at every time-step. In this case, the optimal control inputs can be computed offline then applied to the system in open-loop. Without feedback, however, the controller is incapable of overcoming model error that causes performance to degrade as the time-horizon increases. To counteract this shortcoming, a nonlinear MPC controller can be supplemented with a linear feedback controller based on the local linearization of the nonlinear model about a reference trajectory (see [86, Section 5.4]).

The linearized error dynamics are represented as a discrete-time linear time-variant (LTV) system of the following form:

$$\boldsymbol{\delta}_y[k+1] = A[k]\boldsymbol{\delta}_y[k] + B[k]\boldsymbol{\delta}_u[k] \tag{4.3}$$

where

$$\begin{aligned}
\boldsymbol{\delta}_y[k] =& \boldsymbol{y}[k] - \overline{\boldsymbol{y}}[k] \\
\boldsymbol{\delta}_u[k] =& \boldsymbol{u}[k] - \overline{\boldsymbol{u}}[k] \\
A[k] :=& \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right|_{\substack{\boldsymbol{y}=\overline{\boldsymbol{y}}[k] \\ \boldsymbol{u}=\overline{\boldsymbol{u}}[k]}} \in \mathbb{R}^{n\times n} \\
B[k] :=& \left.\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{y}=\overline{\boldsymbol{y}}[k] \\ \boldsymbol{u}=\overline{\boldsymbol{u}}[k]}} \in \mathbb{R}^{n\times m}
\end{aligned} \tag{4.4}$$

and where at the k^{th} time-step $\overline{\boldsymbol{y}}[k]$ is the reference trajectory and $\overline{\boldsymbol{u}}[k]$ is the control input computed by solving (4.6) or (4.2), $\boldsymbol{\delta}_y[k]$ is the difference between actual state and reference state, while $\boldsymbol{\delta}_u[k]$ is the difference between actual input and reference input. At each time-step k, the optimal input $\boldsymbol{u}^*[k]$ is then given by the sum of the open-loop control input computed by solving the aforementioned NMPC offline, denoted $\overline{\boldsymbol{u}}[k]$, and a feedback term based on the local linearization that is computed online, denoted $\boldsymbol{\delta}_u[k]$,

$$\boldsymbol{u}^*[k] = \overline{\boldsymbol{u}}[k] + \boldsymbol{\delta}_u[k] \tag{4.5}$$

where $\boldsymbol{\delta}_u[k]$ is chosen such that it drives the locally linearized dynamics to zero. In practice, $\boldsymbol{\delta}_u[k]$

Algorithm 6: Koopman-based Discrete Nonlinear MPC (K-NMPC)

Input: Prediction horizon: N_h
Cost matrices: $Q[i], R[i], q[i], r[i]$ for $i = 0, ..., N_h$
Constraint matrices: $E[i], F[i], b[i]$ for $i = 0, ..., N_h$
Model discrete map: $\boldsymbol{T} : Y \times U \to \mathbb{R}^n$

for $k = 0, 1, 2, ...$ **do**
- **Step 1:** Set $\acute{\boldsymbol{y}}[0] = \boldsymbol{y}[k]$)
- **Step 2:** Solve (4.2) to find optimal input $(\boldsymbol{u}[i]^*)_{i=0}^{N_h}$
- **Step 3:** Set $\boldsymbol{u}[k] = u[0]^*$
- **Step 4:** Apply $\boldsymbol{u}[k]$ to the system

end

is computed at each time-step by solving a linear MPC program with structure similar to (4.1).

4.1.2.2 Continuous NMPC

NMPC controllers can also be constructed for continuous-time nonlinear Koopman models. For continuous Koopman-based NMPC, one solves for an optimal function describing the evolution of control inputs over a horizon starting from an instance of time τ of the system according to the the following nonlinear program:

$$\begin{aligned}
\min_{\acute{\boldsymbol{u}} \in L^2([t_0,t_f];\mathbb{R}^m)} \quad & \acute{\boldsymbol{y}}(t_f)^\top Q_f \acute{\boldsymbol{y}}(t_f) + q_f^\top \acute{\boldsymbol{y}}(t_f) \\
& + \int_{t_0}^{t_f} \Big(\acute{\boldsymbol{y}}(t)^\top Q \acute{\boldsymbol{y}}(t) + \acute{\boldsymbol{u}}(t)^\top R \acute{\boldsymbol{u}}(t) + q^\top \acute{\boldsymbol{y}}(t) + r^\top \acute{\boldsymbol{u}}(t) \Big) dt \\
\text{s.t.} \quad & \dot{\acute{\boldsymbol{y}}}(t) = \bar{F}\left(\acute{\boldsymbol{y}}(t), \acute{\boldsymbol{u}}(t)\right), \quad \forall t \in [t_0, t_f] \\
& L(\acute{\boldsymbol{y}}(t), \acute{\boldsymbol{u}}(t)) \leq 0_q, \quad \forall t \in [t_0, t_f] \\
& \acute{\boldsymbol{y}}(t_0) = \boldsymbol{y}(\tau)
\end{aligned} \tag{4.6}$$

where $L^2([t_0, t_f]; \mathbb{R}^m)$ is the space of square integrable functions from $[t_0, t_f]$ to $\mathbb{R}^m$, $G, G_f \in \mathbb{R}^{n \times n}$ and $H \in \mathbb{R}^{m \times m}$ are positive semidefinite matrices, $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^c$ defines state and input constraints where c denotes the total number of imposed constraints, and 0_c is the vector of all 0's with c-elements.

There are commercial software tools designed specifically to solve problems of this type such

as GPOPS-II [83], CasADi [87], SNOPT [88], and Frost [84]. However, just as in the discrete-time case, the time required to solve this problem is usually too large to be used for closed-loop control. Therefore, open-loop control sequences are typically computed offline and feedback strategies based on local linearizations of the dynamics are employed to make these open-loop trajectories more robust to disturbances.

4.1.3 Suboptimal Bilinear MPC

K-MPC and K-NMPC controllers each present different strengths and weaknesses. K-MPC controllers are computationally efficient enough to be updated in real time to make up for modeling error and recover from disturbances, but they are based on linear Koopman realizations which are generally less accurate than nonlinear Koopman realizations of the same system. K-NMPC controllers are based on the more accurate nonlinear Koopman realizations, but cannot adapt to changing conditions because they require solutions to be computed offline.

Luckily, there is another controller which combines the complementary strengths of K-MPC and K-NMPC that is based on bilinear Koopman realizations. Bilinear Koopman realizations generate predictions that are nearly as accurate as nonlinear Koopman realizations (see Section 3.2.1), but they have a structure that is "almost linear". Due to this feature, under some minor assumptions, a bilinear model predictive control problem can be recast as a linear model predictive control problem that can be solved efficiently.

Our proposed Koopman-based bilinear model predictive control (K-BMPC) algorithm, determines a sequence of control inputs over a receding horizon by approximating the solution to the

following optimization problem at each time instance k of the closed-loop operation:

$$\begin{aligned}
\min_{\substack{\{\acute{\boldsymbol{u}}[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{\acute{\boldsymbol{z}}[i]\in\mathbb{R}^N\}_{i=0}^{N_h}}} \quad & \acute{\boldsymbol{z}}[N_h]^T Q[N_h]\acute{\boldsymbol{z}}[N_h] + q[N_h]^T\acute{\boldsymbol{z}}[N_h] + \\
& + \sum_{i=0}^{N_h-1}\Big(\acute{\boldsymbol{z}}[i]^T Q[i]\acute{\boldsymbol{z}}[i] + \acute{\boldsymbol{u}}[i]^T R[i]\acute{\boldsymbol{u}}[i] + \\
& + q[i]^T\acute{\boldsymbol{z}}[i] + r[i]^T\acute{\boldsymbol{u}}[i]\Big) \qquad (4.7) \\
\text{s.t.} \quad & \acute{\boldsymbol{z}}[i+1] = \hat{A}\acute{\boldsymbol{z}}[i] + \hat{B}\acute{\boldsymbol{u}}[i] + \sum_{j=1}^{m}\hat{H}_j\acute{\boldsymbol{z}}[i]\acute{u}_j[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& E[i]\acute{\boldsymbol{z}}[i] + F[i]\acute{\boldsymbol{u}}[i] \leq b[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& \acute{\boldsymbol{z}}[0] = \boldsymbol{g}(\boldsymbol{y}[k])
\end{aligned}$$

where $N_h \in \mathbb{N}$ is the prediction horizon, $Q[i] \in \mathbb{R}^{N\times N}$ and $R[i] \in \mathbb{R}^{m\times m}$ are positive semidefinite matrices, and where each time the program is called, the predictions are initialized from the current lifted state $\boldsymbol{g}(\boldsymbol{y}[k])$. 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. This is identical to the linear MPC optimization problem (4.1) except the dynamics constraint (highlighted in red) now enforces that the solution be consistent with a bilinear Koopman realization rather than a linear one. Unfortunately, this set of constraints renders the problem non-convex, so it may not be possible to solve it quickly enough for real-time control.

The optimization problem can be made convex by replacing the dynamics constraints with linear approximations constructed by replacing $\acute{\boldsymbol{z}}[i]$ in the bilinear terms with $\acute{\boldsymbol{z}}[0]$, i.e.

$$\acute{\boldsymbol{z}}[i+1] = \hat{A}\acute{\boldsymbol{z}}[i] + \hat{B}\acute{\boldsymbol{u}}[i] + \sum_{j=1}^{m}\hat{H}_j\acute{\boldsymbol{z}}[0]\acute{u}_j[i], \quad \forall\{i\}_{i=0}^{N_h-1} \qquad (4.8)$$

$$= \hat{A}\acute{\boldsymbol{z}}[i] + \left(\hat{B} + \begin{bmatrix}\hat{H}_1\acute{\boldsymbol{z}}[0] & \cdots & \hat{H}_m\acute{\boldsymbol{z}}[0]\end{bmatrix}\right)\acute{\boldsymbol{u}}[k], \quad \forall\{i\}_{i=0}^{N_h-1} \qquad (4.9)$$

The resulting linear dynamics approximate the evolution of the system in a neighborhood of the initial lifted state $\acute{\boldsymbol{z}}[0]$. This introduces prediction error, but turns (4.7) into a convex quadratic

Algorithm 7: Koopman-based Suboptimal Bilinear MPC (K-BMPC)

Input: Prediction horizon: N_h
Cost matrices: $Q[i], R[i], q[i], r[i]$ for $i = 0, ..., N_h$
Constraint matrices: $E[i], F[i], b[i]$ for $i = 0, ..., N_h$
Model matrices: $\hat{A}, \hat{B}, \{\hat{H}_j\}_{j=1}^{m}$

for $k = 0, 1, 2, ...$ **do**
  Step 1: Set $\acute{\boldsymbol{z}}[0] = g(y[k])$
  Step 2: Solve (4.1) 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

end

program which can be solved quickly enough to be updated at every time-step:

$$
\begin{aligned}
\min_{\substack{\{\acute{\boldsymbol{u}}[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{\acute{\boldsymbol{z}}[i]\in\mathbb{R}^N\}_{i=0}^{N_h}}} \quad & \acute{\boldsymbol{z}}[N_h]^T G[N_h]\acute{\boldsymbol{z}}[N_h] + g[N_h]^T\acute{\boldsymbol{z}}[N_h] + \\
& + \sum_{i=0}^{N_h-1} \Big(\acute{\boldsymbol{z}}[i]^T G[i]\acute{\boldsymbol{z}}[i] + \acute{\boldsymbol{u}}[i]^T H[i]\acute{\boldsymbol{u}}[i] + \\
& + g[i]^T\acute{\boldsymbol{z}}[i] + h[i]^T\acute{\boldsymbol{u}}[i]\Big) \qquad (4.10) \\
\text{s.t.} \quad & \acute{\boldsymbol{z}}[i+1] = \hat{A}\acute{\boldsymbol{z}}[i] + \left(\hat{B} + \begin{bmatrix}\hat{H}_1\acute{\boldsymbol{z}}[0] & \cdots & \hat{H}_m\acute{\boldsymbol{z}}[0]\end{bmatrix}\right)\acute{\boldsymbol{u}}[k], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& E[i]\acute{\boldsymbol{z}}[i] + F[i]\acute{\boldsymbol{u}}[i] \leq b[i], \quad \forall\{i\}_{i=0}^{N_h-1} \\
& \acute{\boldsymbol{z}}[0] = \boldsymbol{g}(\boldsymbol{y}[k])
\end{aligned}
$$

The main difference between the K-BMPC controller and the other model predictive control algorithms is that it does not generate optimal solutions with respect to the model upon which it is based. Instead, the solutions it generates are optimal with respect to a linear approximation of the actual bilinear model predictive control problem. It is important to note, however, that all Koopman realizations constructed from data are mere approximations of the true dynamics of a system. Hence, even the optimal solutions to the model predictive control problems (4.1), (4.2), and (4.6) are not necessarily optimal with respect to the true dynamics of the system. For this reason, in practice K-BMPC may outperform K-MPC and K-NMPC, despite its suboptimality.

4.2 Application: Trajectory Following with Planar 3-Link Arm

To highlight the relative strengths and weaknesses of the Koopmans-based control techniques described in the previous section, we applied them to the simulated 3-link planar arm system from 3.2 (shown in Fig. 3.3). A K-MPC controller, K-BMPC controller, and discrete K-NMPC controller was constructed from the linear, bilinear, and nonlinear model realizations identified in Section 3.2 for $\rho = 3$. Each controller was then employed to perform the same trajectory following task.

4.2.1 Control Performance Comparison

Each controller computed solutions over a $N_h = 10$ step horizon, had identical cost functions, and no state or input polyhedral constraints. The desired task was to move the end effector of the arm along a planar reference trajectory. Therefore, a cost function was chosen that penalizes the distance between the actual end effector coordinates (α_3, β_3) and the desired coordinates $(\alpha_3^{\text{ref}}, \beta_3^{\text{ref}})$ at each time-step:

$$\begin{aligned}\text{Cost}\left(\{\acute{\boldsymbol{y}}[i]\}_{i=0}^{N_h}, \{\acute{\boldsymbol{u}}[i]\}_{i=0}^{N_h}\right) =& 100\left(C^{\text{ref}}\acute{\boldsymbol{y}}[N_h] - \begin{bmatrix}\alpha_3^{\text{ref}}[N_h]\\ \beta_3^{\text{ref}}[N_h]\end{bmatrix}\right)^{\top}\left(C^{\text{ref}}\acute{\boldsymbol{y}}[N_h] - \begin{bmatrix}\alpha_3^{\text{ref}}[N_h]\\ \beta_3^{\text{ref}}[N_h]\end{bmatrix}\right) + \\ &+ \sum_{i=0}^{N_h-1} 10\left(C^{\text{ref}}\acute{\boldsymbol{y}}[i] - \begin{bmatrix}\alpha_3^{\text{ref}}[i]\\ \beta_3^{\text{ref}}[i]\end{bmatrix}\right)^{\top}\left(C^{\text{ref}}\acute{\boldsymbol{y}}[i] - \begin{bmatrix}\alpha_3^{\text{ref}}[i]\\ \beta_3^{\text{ref}}[i]\end{bmatrix}\right) + \\ &+ (1\times 10^{-3})\,\acute{\boldsymbol{u}}[i]^{\top}\begin{bmatrix}3 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix}\acute{\boldsymbol{u}}[i]\end{aligned} \tag{4.11}$$

where C^{ref} is a selection matrix that isolates the coordinates of the end effector which are the last two components of $\acute{\boldsymbol{y}}[i]$:

$$C^{\text{ref}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.12}$$

Notice that (4.11) also includes a term that penalizes the magnitude of the input to distinguish between multiple robot configurations that all achieve the desired end effector placement.

Control trials were conducted in simulation. At each time-step the current output of the system is measured and used to initialize the MPC optimization problem. Once a solution is computed, the system is simulated forward one time-step ($T_s = 0.05$ seconds) under the optimal input. This procedure is repeated until the end of the reference trajectory is reached.

The reference trajectory chosen traces out the shape of a block letter M over a time period of 15 seconds, starting from the robot's hanging position. Fig. 4.1 shows the path of the end effector using each of the controllers, and Fig. 4.2 displays the mean tracking error and mean computation time over all time-steps. The tracking error at each time-step is quantified as the Euclidean distance between the actual and desired end effector locations. The computation time per iteration is the amount of time it takes to solve the MPC optimization problem and does not include the time to simulate the response of the system. All three trials were run on a computer with 64 GB RAM and a 2.4 GHz CPU.

4.2.2 Discussion

It is clear by inspection of Fig. 4.1 that the K-MPC controller performs very poorly. This is confirmed quantitatively in Fig. 4.2 which shows that its mean tracking error is more than 15$\times$ larger than that of the other controllers. This poor performance can be attributed to the inaccuracy of the linear Koopman model realization upon which it is based, which was documented in Section 3.2. The K-BMPC and K-NMPC controllers track the desired trajectory with much greater fidelity, reflecting the greater accuracy of the bilinear and nonlinear model realizations upon which they are

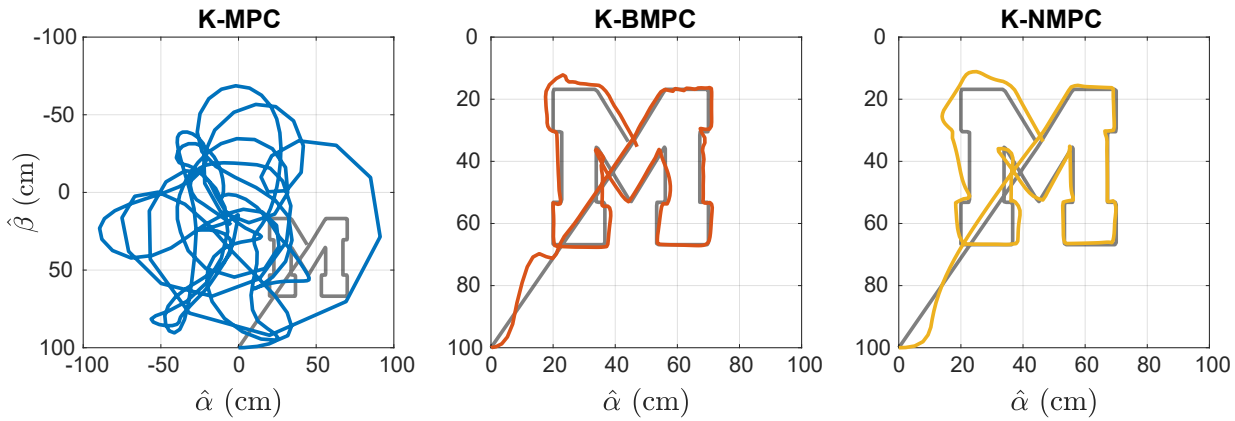


Figure 4.1: The end effector trajectories generated by each Koopman-based model predictive controller superimposed over the same reference trajectory, shown in grey.

based.

In the previous section, we asserted that K-NMPC is much less computationally efficient than the other controllers, and that is confirmed by the results of this experiment. As seen in Fig. 4.2, the mean computation time for K-NMPC is more than 500$\times$ larger than that of the other two controllers. This computation time greatly exceeds the 50 ms duration of a single time-step, making it incompatible with closed-loop operation. Hence, if this robot were a real physical system, the control inputs would have to be computed offline.

Based on the results of this experiment, only K-BMPC would be a viable closed-loop controller for this system. Its mean computation time is much less than a single time-step, and despite the suboptimality of its solutions, its mean tracking error is nearly equivalent to that of K-NMPC. Roughly speaking, K-MPC fast but inaccurate, K-NMPC is accurate but slow, and K-BMPC is both fast and accurate.

The benefits of using a bilinear Koopman realization to model and control a robot are illustrated by this example, but the difference in performance between K-BMPC and K-MPC may not always be so pronounced. Some systems are described very accurately by linear Koopman models, in which case a K-MPC controller may be sufficient. It is up to the control designer to determine which realization and controller is best suited for each particular system.

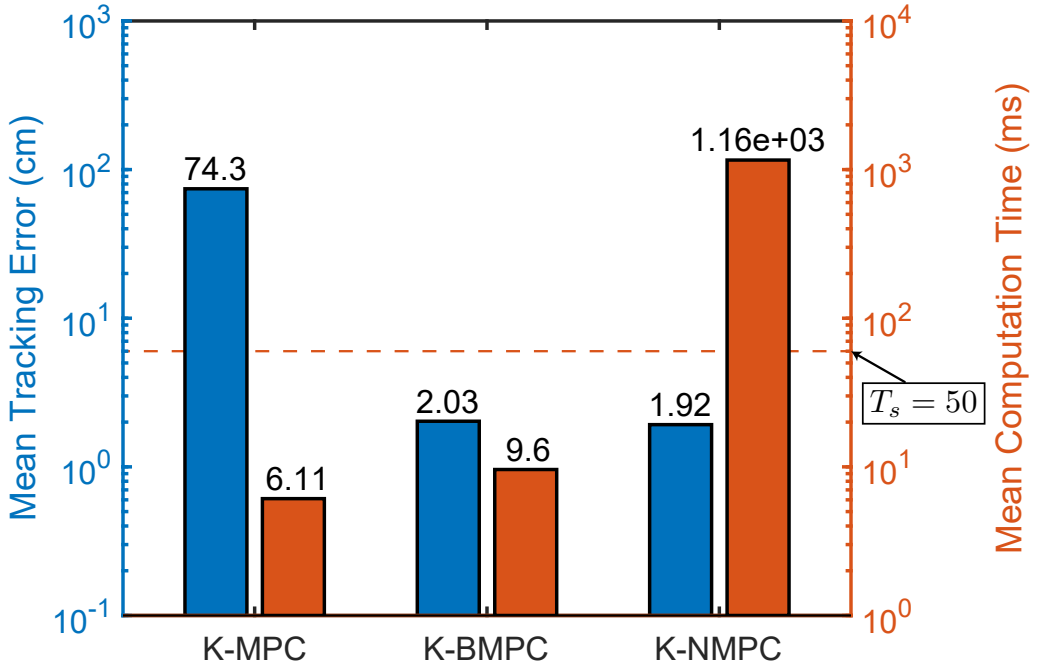


Figure 4.2: The mean tracking error (blue) and the mean computation time (orange) for each controller plotted on a logarithmic scale. The K-BMPC controller has a mean tracking error comparable to K-NMPC and a mean computation time comparable to K-MPC, proving it to be both accurate and computationally efficient.

4.3 Application: Trajectory Following with Soft Robot Arm

This section describes a soft robot platform and the set of experiments used to demonstrate the efficacy of the Koopman-based controllers described in Section 4.1 applied to a real soft robotic system. Footage of the soft robot performing several trajectory following tasks can be found in a supplementary video file[1], and code used to construct the Koopman-based models and controllers for these experiments can be found in a publicly accessible repository[2].

4.3.1 Robot Description: Soft Arm with a Laser Pointer

We experimentally evaluated the Koopman model predictive control approach on a soft robot arm performing the task of drawing shapes with a laser pointer, similar to the handwriting task described in [89]. The robot is a suspended soft arm with a laser pointer attached to the end effector (see Fig. 4.3). The laser dot is projected onto a 50 cm $\times$ 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 or PAMs (also known as McKibben actuators [48]) adhered to a central foam spine by latex rubber bands (see Fig. 4.3). 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

[1] https://youtu.be/FgZmlQh3Ffc

[2] https://github.com/ramvasudevan/soft-robot-koopman

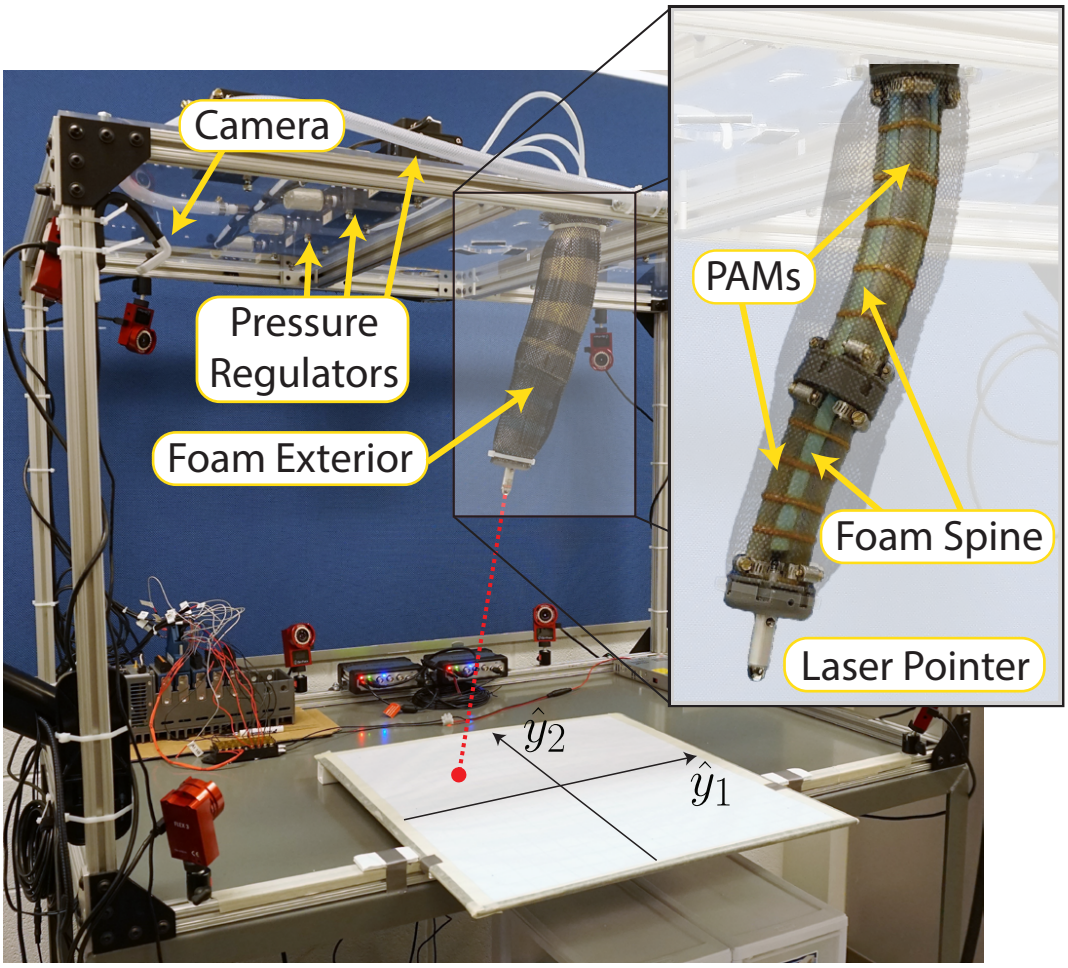


Figure 4.3: The soft robot consists of two bending segments encased in a foam exterior with a laser pointer attached to the end effector. A set of three pressure regulators is used to control the pressure inside of the pneumatic actuators (PAMs), and a camera is used to track the position of the laser dot.

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.

4.3.2 Characterization of Stochastic Behavior

Most mechanical systems demonstrate stochastic behavior (i.e. when an identical input and state produces a different output) to some extent. Stochastic behavior is characteristic of electronic pressure regulators, which can limit the precision of pneumatically driven soft robotic systems and undermine the predictive capability of models.

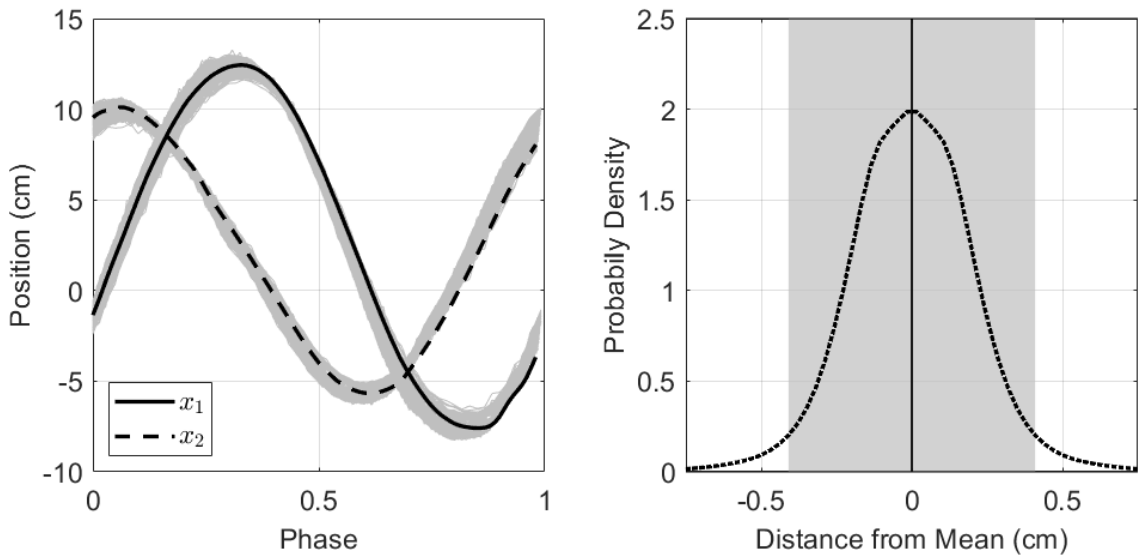


Figure 4.4: The left plot shows the average response of our soft robot system over a single period when the sinusoidal inputs of varying frequencies described by (4.13) are applied. All of the particular responses are subimposed in light grey. The right plot shows the distribution of trajectories about the mean, with all distances within two standard deviations (0.43 cm) highlighted in grey. The width of the distribution illustrates how it is possible for identical inputs to produce outputs that vary by almost 1 cm.

We quantified the stochastic behavior 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

$$\boldsymbol{u}[k] = \begin{bmatrix} 6\sin(\frac{2\pi}{T}kT_s) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{T}{3}) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{2T}{3}) + 3 \end{bmatrix} \tag{4.13}$$

for periods of $T = 6, 7, 8, 9, 10, 11, 12$ seconds and a sampling time of $T_s = 0.083$ seconds 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. 4.4 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 stochasticity will limit the tracking performance of the system, independent of the employed controller.

4.3.3 Data Collection and Model Identification

Data for constructing models was collected over 12 trials lasting approximately 5 minutes 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 zero and ten 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 $T_s = 0.083$ seconds with a zero-order-hold between samples

$$u_i[k] = \frac{(\Upsilon_{i,j+1} - \Upsilon_{i,j})}{T_u}\left(kT_s + \frac{(i-1)T_u}{3}\right) + \Upsilon_{i,j} \tag{4.14}$$

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

Four models were fit from data: a linear state-space model using the subspace method [90], a linear Koopman model using the approach from Section 3.1.3.1, a nonlinear Koopman model using the approach from Section 3.1.3.4, and a Nonlinear Autoregressive with External Input (NARX) neural network model identified using the Levenberg-Marquardt backpropogation algorithm. Each of theses models was trained from the same set of randomly generated data just once, independent of any specific task.

The linear state-space model provides a baseline for comparison and was identified using the MATLAB System Identification Toolbox [71]. It is a four dimensional linear state-space model expressed in observer canonical form.

The linear Koopman model was identified using a set of $45,103$ snapshot pairs $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}\}$ that

incorporate a single delay $d = 1$:

$$\boldsymbol{a}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k)} \\ \boldsymbol{y}^{(k-1)} \\ \boldsymbol{u}^{(k-1)} \end{bmatrix}, \qquad \boldsymbol{b}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k+1)} \\ \boldsymbol{y}^{(k)} \\ \boldsymbol{u}^{(k)} \end{bmatrix} \tag{4.15}$$

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, (3.48) 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. 4.5 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.

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

$$\boldsymbol{a}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k)} \\ \boldsymbol{u}^{(k)} \end{bmatrix}, \qquad \boldsymbol{b}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k+1)} \\ \boldsymbol{u}^{(k)} \end{bmatrix} \tag{4.16}$$

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, (3.48) 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 had 10 hidden neurons, used sigmoid activation functions, and was trained using the MATLAB Neural Network Toolbox [71].

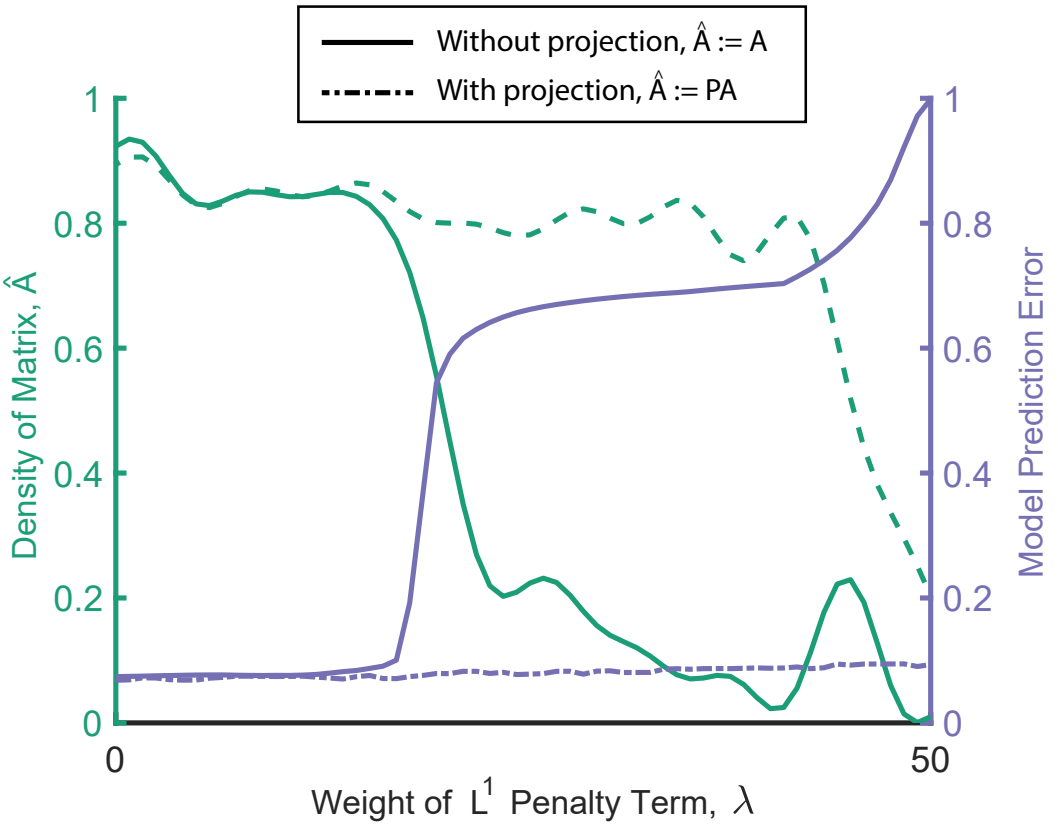


Figure 4.5: As the weight of λ (the L^1 penalty term in (3.48)) increases, the density of the lifted system matrix $\hat{A}$ decreases compared to the least-squares solution which occurs at $\lambda = 0$. If the projection operator P (defined in (3.51)) is not applied (shown by the the solid lines), 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. Here, a model prediction error of 0 denotes perfect tracking, and a model prediction error of 1 denotes the tracking error of a prediction that remains at the origin for all time.

Table 4.1: Average Prediction Error Under Sinusoidal Inputs (cm)

	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

4.3.4 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 (4.13). This comparison data was 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 4.1 displays the average Euclidean distance between the predicted laser dot position and the actual position at each point in time, and Fig. 4.6 shows the tracking performance for the case when the inputs have period $T = 6$ seconds. These results illustrate that the Koopman models generate more accurate predictions than the state space and neural network models with the nonlinear Koopman model's predictions being the best.

4.3.5 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 controller based on the linear state-space model,

K-MPC : Linear MPC controller based on the linear Koopman model,

K-NMPC : Nonlinear MPC controller based on the nonlinear Koopman model,

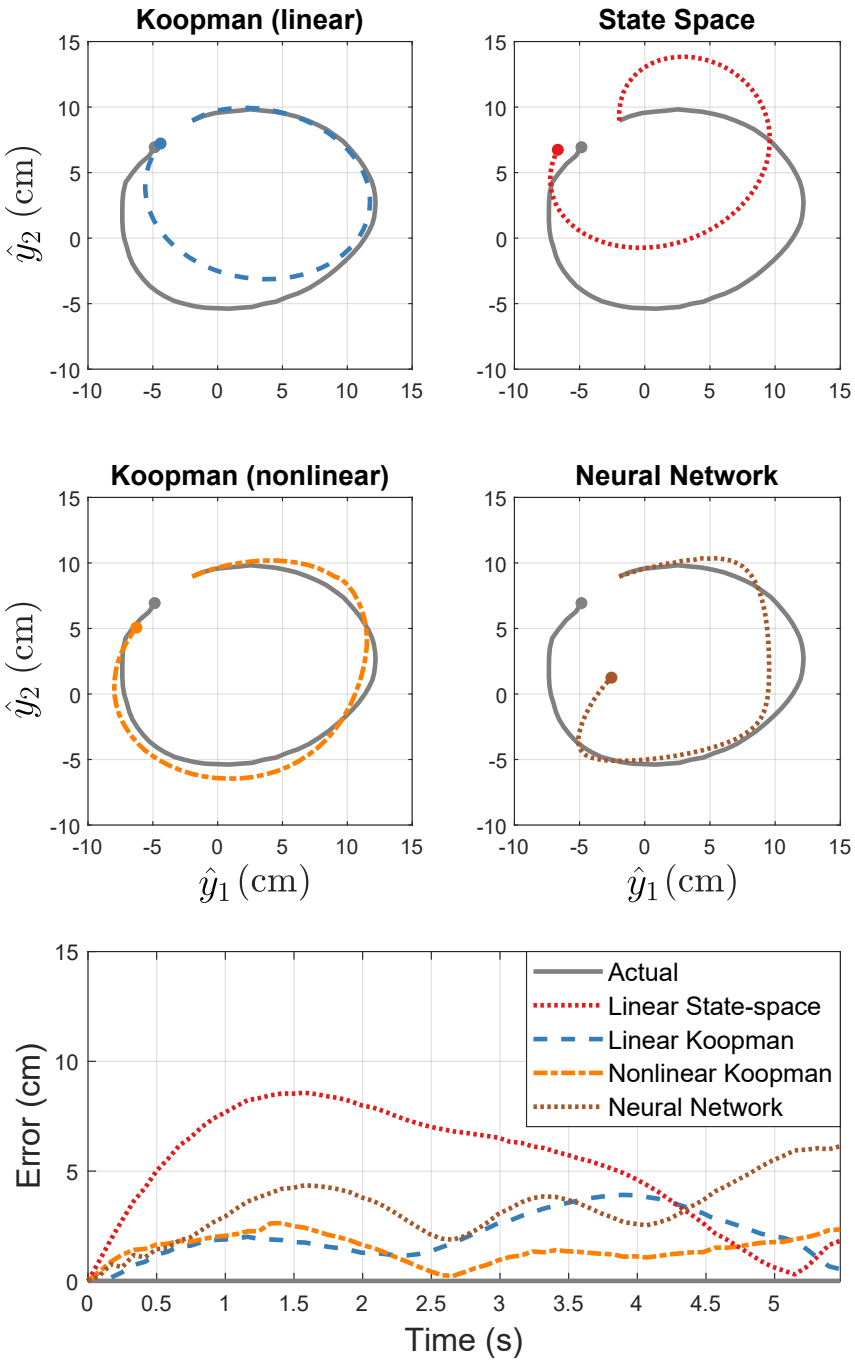


Figure 4.6: The actual response and the model predictions for the robot with the sinusoidal inputs described in (4.13) with period $T = 6$ seconds applied. The left plot shows the actual trajectory of the laser dot along with model predictions. The error displayed on the bottom plot is defined as the Euclidean distance between the predicted laser dot position and the actual position at each point in time.

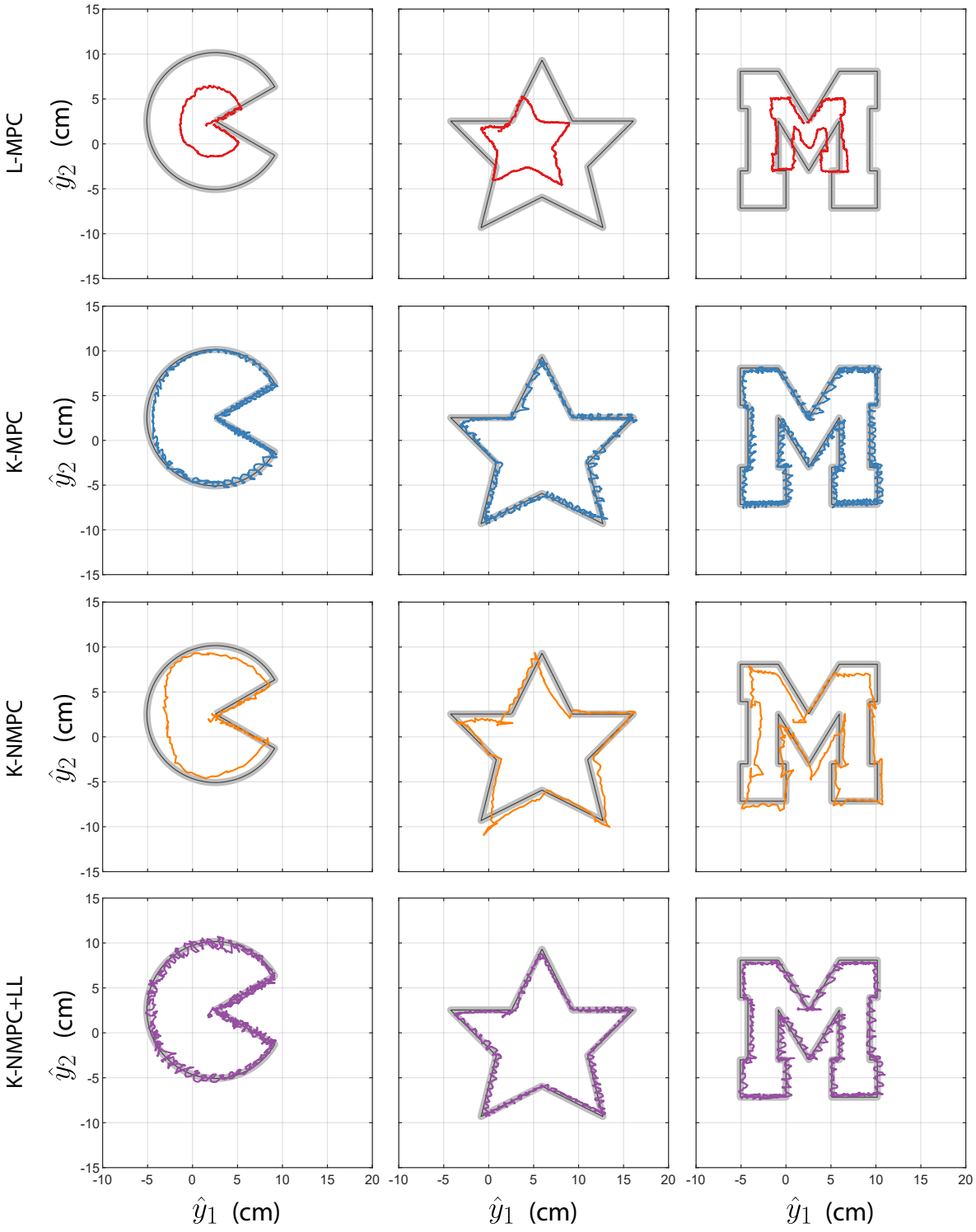


Figure 4.7: The results of the L-MPC controller (row 1, red), the K-MPC controller (row 2, blue), the K-NMPC controller (row 3, orange), and the K-NMPC+LL controller (row 4, purple) in performing trajectory following tasks 1-3. The reference trajectory for each task is subimposed in black as well as a grey buffer with width equal to two standard deviations of the noise probability density shown in Fig. 4.4.

K-NMPC+LL : Nonlinear MPC controller based on the nonlinear Koopman model plus a linear feedback term.

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

The two controllers based on linear models (L-MPC , K-MPC) both solve an optimization problem in the form of (4.1) at each time step using the Gurobi Optimization software [91]. They run in closed-loop at 12 Hz, feature an MPC horizon of 2 seconds ($N_h = 24$), and a cost function that penalizes deviations from a reference trajectory $\boldsymbol{y}^{\text{ref}}$ over the horizon with both a running and terminal cost:

$$\begin{aligned}\text{Cost}\left(\{\acute{\boldsymbol{y}}[i]\}_{i=0}^{N_h}\right) =& 100\left(\boldsymbol{y}[N_h]-\boldsymbol{y}^{\text{ref}}[N_h]\right)^\top\left(\boldsymbol{y}[N_h]-\boldsymbol{y}^{\text{ref}}[N_h]\right)+\\ &+\sum_{i=0}^{N_h-1}0.1\left(\boldsymbol{y}[i]-\boldsymbol{y}^{\text{ref}}[i]\right)^\top\left(\boldsymbol{y}[i]-\boldsymbol{y}^{\text{ref}}[i]\right)\end{aligned} \tag{4.17}$$

In the L-MPC case, $\boldsymbol{y}[i] = C_L\boldsymbol{x}_L[i]$ where $\boldsymbol{x}_L$ 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, $\boldsymbol{y}[i] = C\boldsymbol{z}[i]$, where C is defined as in (3.35).

The two controllers based on the nonlinear Koopman model (K-NMPC , 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 (4.6) offline, with $Q = 100 \times I_{2\times2}$, $R = I_{3\times3}$. The K-NMPC controller 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 (4.5) 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.

Table 4.2: Average Error in Trajectory Following Tasks (cm)

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 (open-loop)	0.95	0.96	0.99	0.96	0.02
K-NMPC (with feedback)	0.43	0.37	0.39	0.40	0.03

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: Pacman (90 seconds)
2. Task 2: Star (120 seconds)
3. Task 3: Block letter M (180 seconds)

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:

$$\text{Error} = \frac{\sum_{i=0}^{N\text{steps}} \sqrt{\left(\boldsymbol{y}[i] - \boldsymbol{y}^{\text{ref}}[i]\right)^{\top} \left(\boldsymbol{y}[i] - \boldsymbol{y}^{\text{ref}}[i]\right)}}{N_{\text{steps}}} \tag{4.18}$$

where N_{steps} denotes the total number of time steps in a trial. The performances of all four controllers in completing Tasks 1, 2, and 3 are shown visually in Fig. 4.7, and the error for each trial is shown in Table 4.2.

The K-NMPC open-loop control inputs took 2.80, 11.57, and 15.82 hours to compute offline for Tasks 1, 2 and 3, respectively, using GPOPS-II on a high-end computer setup with 1 TB RAM and 144 CPUs running at 2.4 GHz. The L-MPC and K-MPC controllers computed control inputs over a 2 second receding horizon in less than 0.083 seconds on a computer with 16 GB RAM and a 3.6 GHz CPU, requiring no offline computations.

4.3.6 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 3 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 resulted 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 instance of them.

Considering that the standard deviation under repeated inputs as described in Section 4.3.2 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 4.3×10^{-3} %. Normalized the same way, the average errors exhibited by the K-MPC and K-NMPC+LL controllers are 9.2×10^{-3} % and 8.0×10^{-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 6.9×10^{-2} %, a full order of magnitude larger than that of the other controllers.

The poor performance of L-MPC 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 second horizon (shown in Table 4.1).

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 hours) that its solution had to be be computed offline, whereas the K-MPC optimization problem could be repeatedly solved in less than 0.083 seconds over a 2 second 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 makes it much more reliable in the presence of external disturbances, which is a topic that could be explored in future work.

CHAPTER 5

Model-based Control Under Loading

One of the primary benefits of soft robots is that they can safely interact with humans. This makes them desirable for human-assistive tasks, many of which require object manipulation. Unfortunately, interacting with objects significantly modifies the dynamics of soft robots since soft materials exhibit substantial non-localized deformation under loading. This poses a challenge for model-based control, which relies on an accurate dynamical model to choose suitable control inputs for a given task. Thus, for a controller to achieve consistent performance in tasks that involve variable loading conditions, it must account for the loading induced changes in the underlying system dynamics.

This chapter presents a Koopman-based framework that explicitly incorporates loading conditions into a dynamical system model to enable real-time control design. By incorporating loads into the model as states, this approach is able to estimate loading online via an observer within the control loop (see Fig. 5.3). This observer infers the most likely value of the loading condition given a series of input-output measurements. The knowledge that is gained in this process enables consistent control performance under a wide range of loading conditions. In fact, the idea of estimating loading conditions has been explored for rigid-bodied robots in the past [92, 93], but has not been explored for robots that experience continuous deformations under load. By using our proposed Koopman-based approach, we are able to transfer these rigid-body results into the world of soft robots.

In Section 5.1, we describe how to incorporate loading conditions into a Koopman model

and how to construct an observer to estimate them. In Section 5.2 we apply this approach to the planar 3-link arm from previous chapters with external loads applied, and demonstrate that online load estimation greatly improves controller performance under variable loading conditions. In Section 5.3, we demonstrate real-time, fully autonomous control of a pneumatically actuated soft continuum manipulator using this approach. In several validation experiments our controller proves itself to be more accurate and more precise across various payloads than several other model-based controllers which do not incorporate loading. This experiment amounts to the first implementation of a closed-loop controller that explicitly accounts for loading on a soft continuum manipulator and the first demonstration of autonomous pick and place for objects of unknown mass on a soft continuum manipulator.

5.1 Incorporating Loading Conditions into Koopman Models

We desire a way to incorporate loading conditions into our dynamic system model and to estimate them online. We can achieve this by including a parametrization of the loading conditions within the lifted states of a Koopman realization of the system, and then constructing an online observer to estimate them. This strategy utilizes the Koopman model to infer the most likely value of the loading conditions given a receding horizon of past input-output measurements.

The proposed observer estimates the loading condition by solving an optimization problem online, but the computational benefits of utilizing a linear or bilinear Koopman realization described in previous chapters would be negated by adding a computationally inefficient observer into the control loop. Therefore, the proposed load-estimation framework incorporates the load into the system such that the load estimation optimization problem is a convex linear program.

5.1.1 Load Estimation for Linear Models

Let $\boldsymbol{w}[i] \in W \subset \mathbb{R}^p$ be a parametrization of the loading conditions on a system at the i^{th} time-step. For example, $\boldsymbol{w}[i]$ might specify the mass at the end effector of a manipulator arm, or the direction

of gravity, or both. We incorporate the loading condition into the lifted state $\boldsymbol{z}[i]$ using a new lifting function $\boldsymbol{\gamma} : Y \times W \to \mathbb{R}^{(N-m)(p+1)}$ which accepts $\boldsymbol{w}[i]$ as a second input and is defined as:

$$\boldsymbol{z}[i] = \boldsymbol{\gamma}(\boldsymbol{y}[i], \boldsymbol{w}[i]) = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{y}[i]) \\ \boldsymbol{g}(\boldsymbol{y}[i])w_1[i] \\ \vdots \\ \boldsymbol{g}(\boldsymbol{y}[i])w_p[i] \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{g}(\boldsymbol{y}[i]) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{g}(\boldsymbol{y}[i]) \end{bmatrix}}_{\Gamma(\boldsymbol{y}[i])} \begin{bmatrix} 1 \\ \boldsymbol{w}[i] \end{bmatrix} \tag{5.1}$$

where $\boldsymbol{g}$ is the original lifting function (3.22), $\Gamma(\boldsymbol{y}[i]) \in \mathbb{R}^{\left((N-m)(p+1)\right)\times(p+1)}$ is the matrix formed by diagonally concatenating $\boldsymbol{g}(\boldsymbol{y}[i])$ $(p+1)$ times, and $w_j[i]$ denotes the j^{th} element of $\boldsymbol{w}[i]$. We purposely define the lifted state as an affine function of the load condition to ensure that solving for it will be computationally efficient. Note that because this lifting function requires the loading condition as an input, it must also be included in the snapshots $(\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{w}^{(k)})$ to construct a Koopman model that accounts for loading.

Although the loading condition is not measured directly, its value can be inferred based on the system model and past input-output measurements. We construct an observer that estimates its value at the i^{th} time-step, $\boldsymbol{w}[i]$, by solving a linear least-squares problem using data from the N_w previous time-steps. Notice that the output at the i^{th} time-step $\boldsymbol{y}[i]$ can be expressed in terms of the input $\boldsymbol{u}[i-1]$, the output $\boldsymbol{y}[i-1]$, and the load $\boldsymbol{w}[i-1]$ at the previous time-step by combining the linear model equations of (3.19) and then substituting (5.1) for $\boldsymbol{z}[i-1]$,

$$\boldsymbol{y}[i] = C\left(A\boldsymbol{z}[i-1] + B\boldsymbol{u}[j-1]\right) \tag{5.2}$$

$$= CA\Gamma(\boldsymbol{y}[i-1]) \begin{bmatrix} 1 \\ \boldsymbol{w}[i-1] \end{bmatrix} + CB\boldsymbol{u}[i-1]. \tag{5.3}$$

Solving for the best estimate of $\boldsymbol{w}[i-1]$ in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}[i-1]$ yields the following

expression,

$$\begin{bmatrix} 1 \\ \bar{\boldsymbol{w}}[i-1] \end{bmatrix} = \left(CA\Gamma(\boldsymbol{y}[j-1])\right)^{\dagger}\left(\boldsymbol{y}[j] - CB\boldsymbol{u}[j-1]\right) \tag{5.4}$$

where † denotes the Moore-Penrose psuedoinverse.

Under the assumption that the loading is equal to some constant $\tilde{\boldsymbol{w}}$ over the previous N_w time steps, i.e. $\boldsymbol{w}[i] = \tilde{\boldsymbol{w}}$ for $i = j - N_w, \ldots, j$, we can similarly find the best estimate over all N_w timesteps. Since this estimate is based on more data it should be more accurate and more robust to noisy output measurements. We define the following two matrices,

$$\Lambda_A = \begin{bmatrix} CA\Gamma(\boldsymbol{y}[j-1]) \\ \vdots \\ CA\Gamma(\boldsymbol{y}[j-N_w]) \end{bmatrix} \tag{5.5}$$

$$\Lambda_B = \begin{bmatrix} \boldsymbol{y}[j] - CB\boldsymbol{u}[j-1] \\ \vdots \\ \boldsymbol{y}[j-N_w+1] - CB\boldsymbol{u}[k-N_w] \end{bmatrix} \tag{5.6}$$

Then, following from (5.4), the best estimate for $\tilde{\boldsymbol{w}}$ over the past N_w time-steps in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}$, is given by

$$\begin{bmatrix} 1 \\ \bar{\boldsymbol{w}} \end{bmatrix} = \Lambda_A^{\dagger}\Lambda_B, \tag{5.7}$$

where † again denotes the Moore-Penrose psuedoinverse.

5.1.2 Load Estimation for Bilinear Models

The load estimation problem for bilinear realizations is very similar to the linear case. The lifted state $\boldsymbol{z}[i]$ is again defined as in (5.1). Then, the output at the i^{th} time-step $\boldsymbol{y}[i]$ can be expressed in

terms of the input $\boldsymbol{u}[i-1]$, the output $\boldsymbol{y}[i-1]$, and the load $\boldsymbol{w}[i-1]$ at the previous time-step by combining the bilinear model equations of (3.28) and then substituting (5.1) for $\boldsymbol{z}[i-1]$,

$$\boldsymbol{y}[i] = C\left(A\boldsymbol{z}[i-1] + B\boldsymbol{u}[i-1] + \sum_{j=1}^{m} H_j \boldsymbol{z}[i-1]u_j[i-1]\right) \tag{5.8}$$

$$= C\left(A\Gamma(\boldsymbol{y}[i-1]) + \sum_{j=1}^{m} H_j \Gamma(\boldsymbol{y}[i-1])u_j[i-1]\right)\begin{bmatrix} 1 \\ \boldsymbol{w}[i-1] \end{bmatrix} + CB\boldsymbol{u}[i-1] \tag{5.9}$$

Solving for the best estimate of $\boldsymbol{w}[i-1]$ in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}[i-1]$ yields the following expression,

$$\begin{bmatrix} 1 \\ \bar{\boldsymbol{w}}[i-1] \end{bmatrix} = \left(CA\Gamma(\boldsymbol{y}[i-1]) + C\sum_{j=1}^{m} H_j \Gamma(\boldsymbol{y}[i-1])u_j[i-1]\right)^{\dagger}\left(\boldsymbol{y}[j] - CB\boldsymbol{u}[j-1]\right) \tag{5.10}$$

where † denotes the Moore-Penrose psuedoinverse.

Again, under the assumption that the loading condition is equal to some constant $\tilde{\boldsymbol{w}}$ over the previous N_w time steps, we can find the best estimate over all N_w time-steps. We define the following two matrices,

$$\Lambda_A = \begin{bmatrix} CA\Gamma(\boldsymbol{y}[i-1]) + C\sum_{j=1}^{m} H_j\Gamma(\boldsymbol{y}[i-1])u_j[i-1] \\ \vdots \\ CA\Gamma(\boldsymbol{y}[i-N_w]) + C\sum_{j=1}^{m} H_j\Gamma(\boldsymbol{y}[i-N_w])u_j[i-N_w]) \end{bmatrix} \tag{5.11}$$

$$\Lambda_B = \begin{bmatrix} \boldsymbol{y}[j] - CB\boldsymbol{u}[j-1] \\ \vdots \\ \boldsymbol{y}[j-N_w+1] - CB\boldsymbol{u}[k-N_w] \end{bmatrix} \tag{5.12}$$

Then, following from (5.10), the best estimate for $\tilde{\boldsymbol{w}}$ over the past N_w time-steps in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}$, is given by (5.7).

5.1.3 Load Estimation for Nonlinear Models

In contrast to the linear and bilinear cases, nonlinear Koopman realizations advance the output directly, rather than via a lifted state. To incorporate loading conditions into a nonlinear Koopman model, we begin by defining a new lifting function that accepts the loading condition $\boldsymbol{w}[i]$ as an additional input $\boldsymbol{\gamma} : \mathbb{R}^{n\times m\times p} \to \mathbb{R}^{N(p+1)}$, defined as

$$\boldsymbol{\gamma}(\boldsymbol{y}[i], \boldsymbol{u}[i], \boldsymbol{w}[i]) = \begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i]) \\ \boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i]) w_1[i] \\ \vdots \\ \boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i]) w_p[i] \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i]) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i]) \end{bmatrix}}_{\Gamma(\boldsymbol{y}[i], \boldsymbol{u}[i])} \begin{bmatrix} 1 \\ \boldsymbol{w}[i] \end{bmatrix} \tag{5.13}$$

where $\boldsymbol{\psi}$ is the original nonlinear lifting function (3.38), $\Gamma(\boldsymbol{y}[i], \boldsymbol{u}[i]) \in \mathbb{R}^{\left(N(p+1)\right)\times(p+1)}$ is the matrix formed by diagonally concatenating $\boldsymbol{\psi}(\boldsymbol{y}[i], \boldsymbol{u}[i])$ $(p+1)$ times, and $w_j[i]$ denotes the j^{th} element of $\boldsymbol{w}[i]$.

Using this lifting function, the nonlinear discrete flow map $\boldsymbol{T} : Y \times U \times W \to \mathbb{R}^n$ resulting from the system identification technique described in Section 3.1.3.3 will be,

$$\boldsymbol{T}(\boldsymbol{y}[i], \boldsymbol{u}[i], \boldsymbol{w}[i]) = \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \boldsymbol{\gamma}(\boldsymbol{y}[i], \boldsymbol{u}[i], \boldsymbol{w}[i]) \tag{5.14}$$

$$= \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \Gamma(\boldsymbol{y}[i], \boldsymbol{u}[i]) \begin{bmatrix} 1 \\ \boldsymbol{w}[i] \end{bmatrix}. \tag{5.15}$$

Using this expression for the discrete flow map, the output at the i^{th} time-step $\boldsymbol{y}[i]$ can be expressed in terms of the input $\boldsymbol{u}[i-1]$, the output $\boldsymbol{y}[i-1]$, and the load $\boldsymbol{w}[i-1]$ at the previous time-step,

$$\boldsymbol{y}[i] = \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \Gamma(\boldsymbol{y}[i-1], \boldsymbol{u}[i-1]) \begin{bmatrix} 1 \\ \boldsymbol{w}[i-1] \end{bmatrix}. \tag{5.16}$$

Solving for the best estimate of $\boldsymbol{w}[i-1]$ in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}[i-1]$ yields the following

expression,

$$\begin{bmatrix} 1 \\ \bar{\boldsymbol{w}}[i-1] \end{bmatrix} = \left(\begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \Gamma(\boldsymbol{y}[i-1], \boldsymbol{u}[i-1]) \right)^{\dagger} \boldsymbol{y}[i]. \tag{5.17}$$

Then, under the assumption that the loading condition is equal to some constant $\tilde{\boldsymbol{w}}$ over the previous N_w time steps, we can find the best estimate over all N_w time-steps. We define the following two matrices,

$$\Lambda_A = \begin{bmatrix} \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \Gamma(\boldsymbol{y}[i-1], \boldsymbol{u}[i-1]) \\ \vdots \\ \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix} \bar{\mathcal{K}}_{T_s}^{\top} \Gamma(\boldsymbol{y}[i-N_w], \boldsymbol{u}[i-N_w]) \end{bmatrix} \tag{5.18}$$

$$\Lambda_B = \begin{bmatrix} \boldsymbol{y}[j] \\ \vdots \\ \boldsymbol{y}[j-N_w+1] \end{bmatrix} \tag{5.19}$$

Then, following from (5.17), the best estimate for $\tilde{\boldsymbol{w}}$ over the past N_w time-steps in the L^2-norm sense, denoted $\bar{\boldsymbol{w}}$, is given by (5.7).

5.1.4 Online Load Observer

A Koopman model that incorporates loading conditions is perfectly compatible with the model-based controllers described in Chapter 4. In this case, the controller predictions depend on the estimate of the loading conditions $\bar{\boldsymbol{w}}$. This estimate must be periodically updated using the method described in the preceding three sections. By construction, solving (5.7) to estimate the load is computationally efficient because it is a convex least-squares problem. However, for systems with relatively stable loading conditions, it is still inefficient to compute a new load estimate at every time-step. Therefore, we define a load estimation update period N_e as the number of time steps to wait between load estimations. Increasing N_e will likely increase the accuracy of each load

Algorithm 8: Koopman MPC with Load Observer

Input: Prediction horizon: N_h
Koopman Model Realization: Linear (3.19), Bilinear (3.28), or Nonlinear (3.36)

for $k = 0, 1, 2, ...$ **do**
  if $k \mod N_e = 0$ **then**
    Estimate $\bar{\boldsymbol{w}}'[j]$ via (5.7)
    $\bar{\boldsymbol{w}}[k]$ = mean($\bar{\boldsymbol{w}}'[j], \bar{\boldsymbol{w}}[k-1], \ldots, \bar{\boldsymbol{w}}[k-N_r]$)
    $j = j + 1$
  else
    $\bar{\boldsymbol{w}}[k] = \bar{\boldsymbol{w}}[k-1]$
  end
  Step 1: Solve MPC problem to find optimal input $(\boldsymbol{u}[i]^*)_{i=0}^{N_h}$
  Step 2: Set $\boldsymbol{u}[k] = \boldsymbol{u}[0]^*$
  Step 3: Apply $\boldsymbol{u}[k]$ to the system
end

estimate, but will also reduce responsiveness to changes in the loading conditions. To balance accuracy with responsiveness, we update $\bar{\boldsymbol{w}}$ every N_e time steps by setting it equal to the average of the new load estimate and the previous N_r load estimates, where N_r is another user defined constant. Algorithm 7 summarizes the closed-loop operation of a Koopman-based MPC controller with these periodic load estimation updates.

5.2 Application: Planar 3-Link Arm with Loads

In Section 3.2 we introduced a planar 3-link arm system, and in Section 4.2 we illustrated how it could be accurately controlled using an MPC controller based on a bilinear Koopman realization. We now consider the same system, but with loads applied.

The loading condition $\boldsymbol{w}[i] \in \mathbb{R}^2$ is defined as the mass of the end effector and the direction of gravity at the i^{th} time-step. Specifically $w_1[i]$ is equal to the mass of the the end effector, and $w_2[i]$ is equal to the angle between the direction of gravity and the $\hat{\beta}$ coordinate axis, as shown in Fig. 5.1. Note that the load condition defined here does not conform to the colloquial use of the word "load" to mean a force. The framework presented in this chapter is flexible enough to accommodate any sort of external condition that affects the dynamics of the system, even if strictly

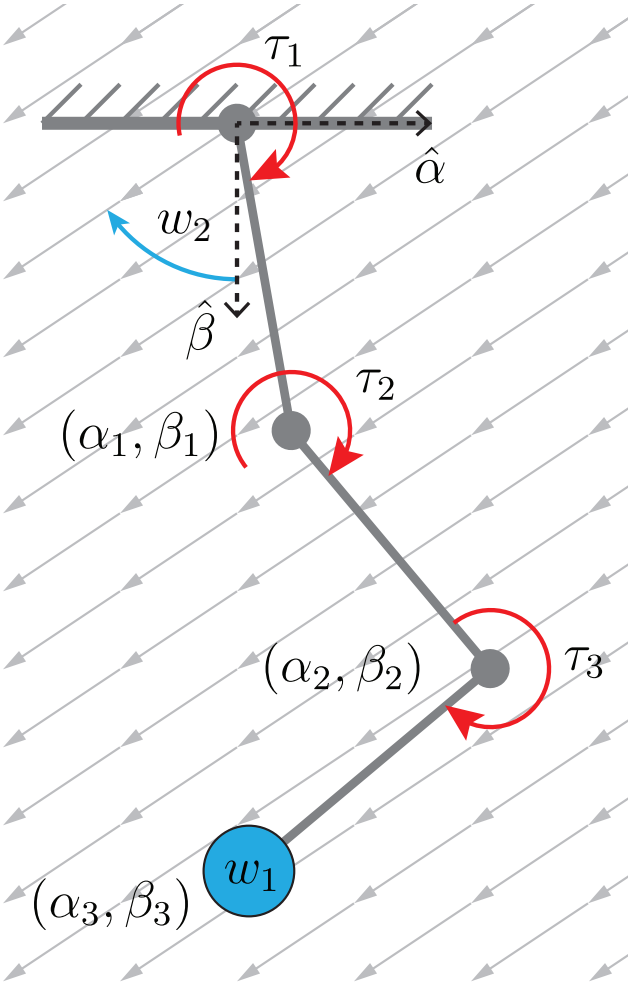


Figure 5.1: Three link planar arm system with joint torques defined as the input and locations of the end of each link defined as the output. A load with mass w_1 kg is attached to the end effector and the direction of gravity with respect to the unit vector $\hat{\beta}$ is described by the angle w_2.

speaking it would not usually be referred to as a load. For this reason, some may prefer to call $\boldsymbol{w}[i]$ an "uncontrolled input" rather than a loading condition.

To showcase the benefits of explicitly accounting for loading conditions in Koopman-based models and controllers, we compare the performance of a controller that does account for loading conditions to one that does not. In Section 4.2 it was concluded that only the MPC controller based on the bilinear Koopman realization (K-BMPC) was a viable closed-loop controller for the system. Therefore, we construct a new bilinear model with lifted state defined as in (5.1) and controller with load observer as described by Algorithm 8, then compare the performance of this controller to that of the K-BMPC controller from Section 4.2. This new controller is denoted

KL-BMPC for "Koopman-based loaded bilinear model predictive controller".

5.2.1 Control Performance Comparison

A bilinear model with loading incorporated into it was identified using data from a collection of 22 trials of length 40 seconds during which randomized "ramp and hold" type inputs were applied. One trial was conducted for every combination of end effector loads in the set $\{0, 1\}$ kg and gravity directions in the set $\{-\frac{\pi}{2} + k\frac{\pi}{10} \big| k = 0, \ldots, 10\}$, for the total of 22 trials. The approximation of the Koopman operator was constructed using the lifted state as defined in (5.1), where the output dependent lifting function $\boldsymbol{g} : Y \to \mathbb{R}^N$ is defined for some $\tilde{\boldsymbol{y}} \in Y \subset \mathbb{R}^6$ as,

$$\boldsymbol{g}(\tilde{\boldsymbol{y}}) = \begin{bmatrix} g_1(\tilde{\boldsymbol{y}}) & \cdots & g_N(\tilde{\boldsymbol{y}}) \end{bmatrix}^\top, \tag{5.20}$$

and the set of functions $\{g_i : Y \to \mathbb{R}\}_{i=1}^N$ is defined as all monomials of the elements of the output up to degree 3 such that $N = (6+3)!/(6!3!) = 84$. The complete lifting function $\boldsymbol{\psi} : Y \times U \times W \to \mathbb{R}^{N(p+1)(m+1)}$ is defined for some $\tilde{\boldsymbol{y}} \in Y$, $\tilde{\boldsymbol{u}} \in U$ and $\tilde{\boldsymbol{w}} \in W$ as,

$$\boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{w}}) = \begin{bmatrix} \boldsymbol{\gamma}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{w}}) \\ \boldsymbol{\gamma}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{w}})\tilde{u}_1 \\ \vdots \\ \boldsymbol{\gamma}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{w}})\tilde{u}_m \\ \tilde{\boldsymbol{u}} \end{bmatrix}. \tag{5.21}$$

$\boldsymbol{\psi}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{w}})$ is 1008 dimensional since $N = 84, p = 2, m = 3$, but using the dimensional reduction technique described in Section 3.1.4.2 this dimension was reduced to 384. The bilinear realization matrices $A, B, \{H_j\}_{j=1}^m$ were identified according to Algorithm 2, without an L^1-penalty term or projection matrix.

This loaded bilinear realization was used to construct a model predictive controller in the form of Algorithm 8, which we refer to by the abbreviation KL-BMPC. The prediction horizon

Table 5.1: Average Tracking Error for Circular Trajectory Following with Load (cm)

	Loading Condition		
	$\boldsymbol{w} = [1, -\frac{\pi}{3}]^\top$	$\boldsymbol{w} = [1, 0]^\top$	$\boldsymbol{w} = [1, \frac{\pi}{3}]^\top$
K-BMPC	13.8	3.8	19.6
KL-BMPC	4.3	1.8	3.8

is $N_h = 10$, and the cost function and constraints are identical to those of the K-BMPC controller from Section 4.2. For the load observer, the load estimation horizon is $N_w = 20$, the number of steps between load estimate updates is $N_e = 10$, and the number of previous estimates to average over is $N_r = 1$. In terms of time units, the load estimation horizon is 1 second and the time between load estimate updates is 0.5 seconds, since the length of one time-step for the system is $T_s = 0.05$ seconds.

The K-BMPC and KL-BMPC controllers were each tasked with moving the end effector along a 15 second circular reference trajectory under the constant loading conditions in the set $\left\{[1, -\frac{\pi}{3}]^\top, [1, 0]^\top, [1, \frac{\pi}{3}]^\top\right\}$. Note that these loading conditions were not among those used in the training trials. The performance of each controller is shown in Fig. 5.2, and the average tracking error over all time-steps for each trial is given in Table 5.1.

5.2.2 Discussion

The KL-BMPC controller achieved a more accurate and consistent control performance across various loading conditions. This showcases the advantage of explicitly incorporating loading conditions into the model and controller. Even though neither controller was provided with explicit knowledge of the loading condition, the KL-BMPC controller was able to improve the accuracy of its model over time by computing load estimates online.

Across all three trials, the average tracking error of the K-BMPC was 2 to 5 times larger than that of the KL-BMPC controller. The smallest difference in performance occurred under the loading condition $\boldsymbol{w} = [1, 0]^\top$. This may be due to the fact that this loading condition is the

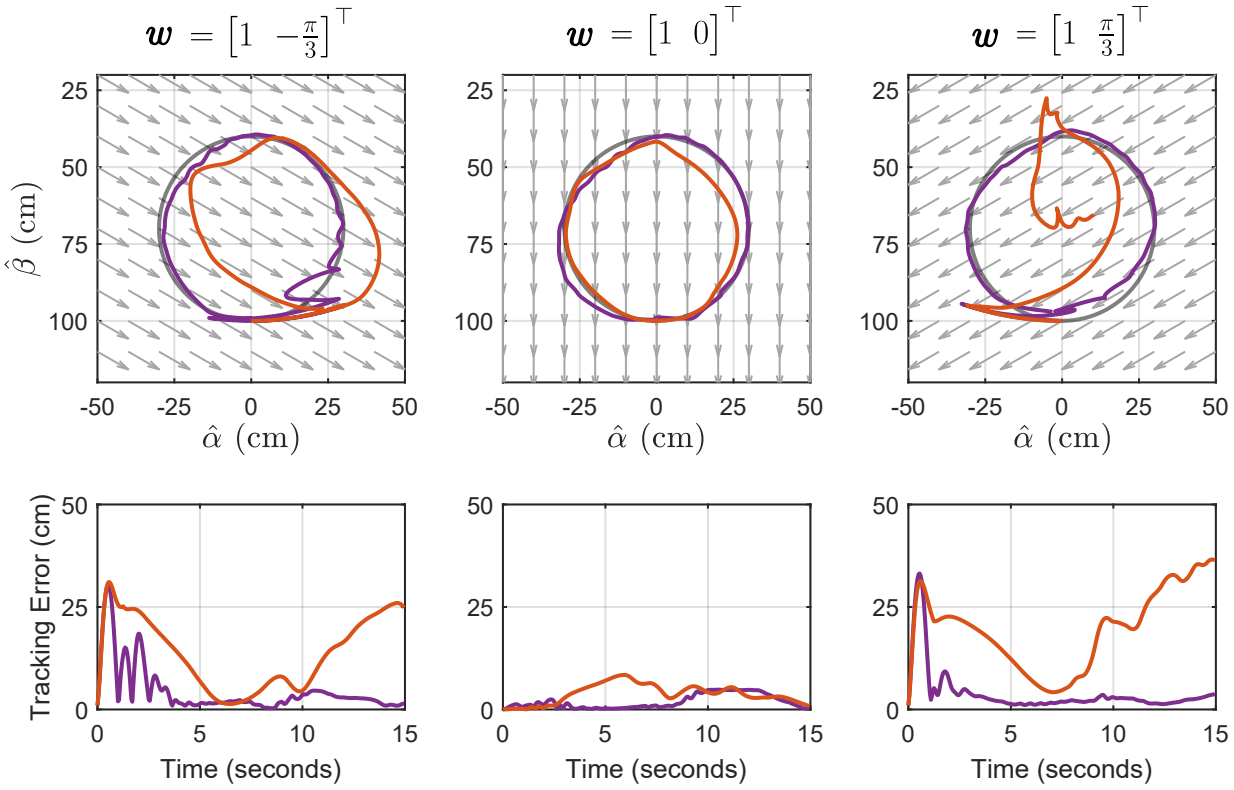


Figure 5.2: The performance of the K-BMPC (red) and KL-BMPC (purple) controllers for a circular trajectory following task under several loading conditions. Each column shows the results for a different direction of the gravitational field, which is illustrated by grey arrows. The top row shows the trajectory of each controller superimposed over the reference trajectory (grey). The bottom row shows the tracking error for each controller over the 15 second trajectory.

most similar to the conditions under which K-BMPC's model was trained (i.e. $\boldsymbol{w} = [0, 0]^\top$).

The error plots in Fig. 5.2 show that at the beginning of each trial, both controllers have nearly identical tracking error. However, as more time passes the load estimates computed by the KL-BMPC controller improve, causing the overall tracking error to decrease significantly. In contrast, the K-BMPC controller is incapable of improving its tracking performance over time because it does not have the capacity to incorporate load information into its model.

5.3 Application: Soft Continuum Manipulator with Payload

Using the modeling and control approach presented in Section 5.1.1, we demonstrate consistent control of a pneumatically-driven soft continuum manipulator arm under various loading conditions. This section describes the soft continuum manipulator and the set of experiments used to demonstrate the efficacy of its controller and load observer. Footage from these experiments can be found in a supplementary video file[1].

5.3.1 System Identification of Soft Robot Arm

To validate the modeling and control approach described in Section 5.1, we applied it to a soft robot arm capable of picking-up objects and moving its end effector in three-dimensional space. The robot, shown in Fig. 5.4, is 70 cm long and has a diameter of 6 cm. It is made up of three pneumatically actuated bending sections and an end effector comprised of a granular jamming vacuum gripper [1]. Each section is actuated by three pneumatic artificial muscles (PAMs) [48] which are adhered to a central spine consisting of an air hose encased in flexible PVC foam tubing. Another much larger sleeve of flexible PVC foam surrounds the actuators, which serves to dampen high frequency oscillations and make the body of the arm softer overall. The air pressure inside the actuators is regulated by 9 Enfield TR-010-g10-s pneumatic pressure regulators that accept $0 - 10$V command signals corresponding to pressures of approximately $0 - 275$ kPa, and are connected to

[1] https://youtu.be/g2yRUoPK40c

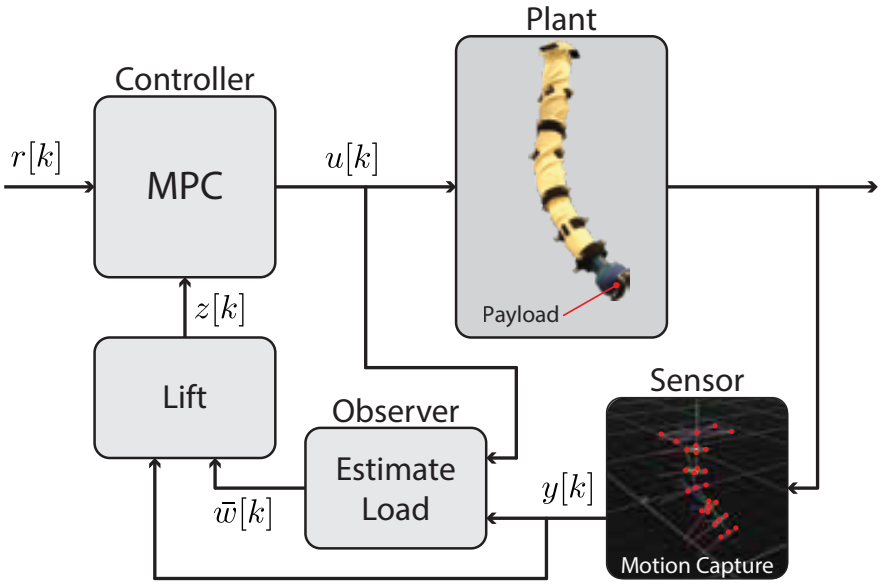


Figure 5.3: A soft continuum manipulator is tasked with following a reference trajectory $\boldsymbol{r}[k]$ while carrying an unknown payload. At each time step a model predictive controller computes the optimal input $\boldsymbol{u}[k]$ to follow the reference trajectory based on a linear Koopman operator model, and the position of the end effector $\boldsymbol{y}[k]$ is measured by a motion capture system. An observer computes a payload estimate $\bar{\boldsymbol{w}}$ based on the previously measured inputs and outputs, which is then incorporated into the state of the model $\boldsymbol{z}[k]$ via a lifting function.

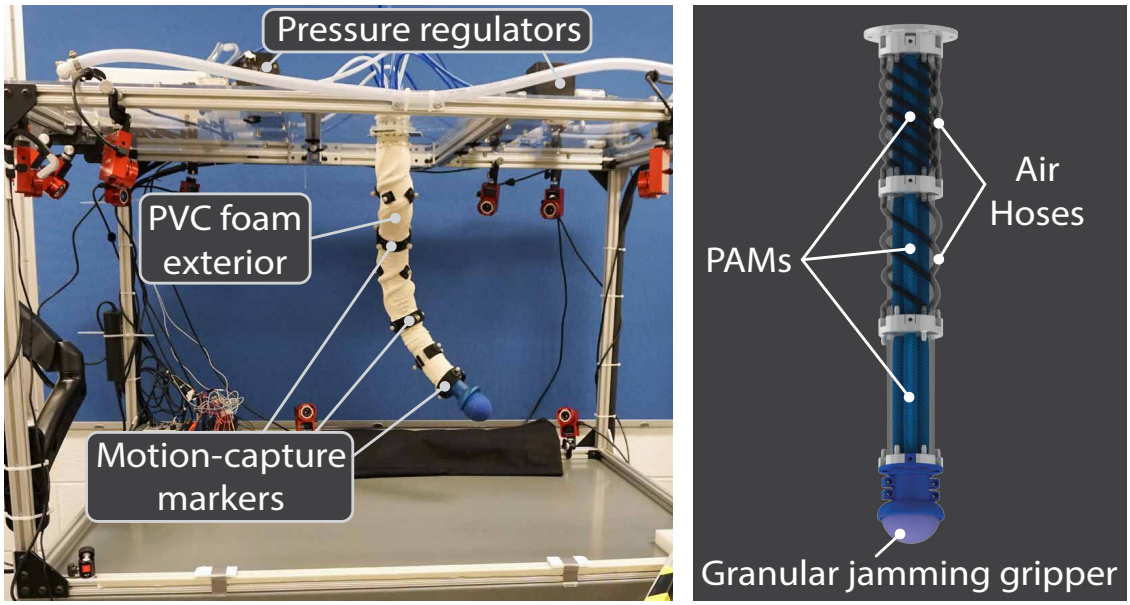


Figure 5.4: The soft robot arm consists of three bending sections, each actuated by three pneumatic artificial muscles (PAMs). The actuators are surrounded by a sleeve of flexible PVC foam, and pressurized air is supplied to the actuators via air hoses that wind around the exterior. The end effector consists of a granular jamming vacuum gripper [1], which is connected to a vacuum pump by a hose that runs along the interior of the arm.

the actuators by air hoses that wrap around the outside of the foam sleeves. The exterior of the arm is covered in retro-reflective markers which are tracked using a commercial OptiTrack motion capture system.

We quantified the stochastic behavior of our soft robot system by observing the variations in output from period-to-period under sinusoidal inputs with a period of 10 seconds and a sampling time of $T_s = 0.083$ seconds with a zero-order-hold between samples. Over 60 periods, the trajectory of the end effector deviated from the mean trajectory by an average of 9.45 mm and with a standard deviation of 7.3 mm. This inherent stochasticity limits the tracking performance of the system, independent of the employed controller.

For the purposes of constructing a dynamic model for the arm, the input was chosen to be the command voltages into the 9 pressure regulators and at each instance in time was restricted to $[0, 10]^9$. The output was chosen to live in $\mathbb{R}^9$ and corresponds to the positions of the ends of each of the 3 bending sections in Cartesian coordinates with the last 3 coordinates corresponding to the end effector position. The parametrization of the loading condition lives in $\mathbb{R}_+$ and is chosen to be the mass of the object held by the gripper.

Data for constructing models was collected over 49 trials lasting approximately 10 minutes each. A randomized "ramp and hold" type input and a load from the set $\{0, 50, 100, 150, 200, 250, 300\}$ grams was applied during each trial to generate a representative sampling of the system's behavior over its entire operating range.

Three models were fit from the data: a linear state-space model using the subspace method [90], a linear Koopman model that *does not* take loading into account using the approach described in Section 3.1.2, and a linear Koopman model that *does* incorporate loading using the approach from Section 5.1.1. Each of theses models was fit using the same set of $325,733$ randomly generated data points just once, independent of any specific task.

The linear state-space model provides a baseline for comparison and was identified using the MATLAB System Identification Toolbox [71]. This model is a 9 dimensional linear state-space model expressed in observer canonical form.

The first Koopman model (without loading) was identified on a set of $K = 325,732$ snapshot pairs $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}\}_{k=1}^{K}$ that incorporate a single delay $d = 1$:

$$\boldsymbol{a}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k)} \\ \boldsymbol{y}^{(k-1)} \\ \boldsymbol{u}^{(k-1)} \end{bmatrix}, \qquad \boldsymbol{b}^{(k)} = \begin{bmatrix} \boldsymbol{y}^{(k+1)} \\ \boldsymbol{y}^{(k)} \\ \boldsymbol{u}^{(k)} \end{bmatrix}. \tag{5.22}$$

Note that the dimension of each snapshot is $2n + m = 2(9) + 9 = 27$ due to the inclusion of the delay, and we denote by $\boldsymbol{y}^d[k] \in \mathbb{R}^{27}$ one of these outputs which has delays included at some time k. The the lifting function $\boldsymbol{\psi} : \mathbb{R}^{27\times 9} \to \mathbb{R}^{111}$ was defined as

$$\boldsymbol{\psi}(\boldsymbol{y}^d[k], \boldsymbol{u}[k]) = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{y}^d[k]) \\ \boldsymbol{u}[k] \end{bmatrix} \tag{5.23}$$

where the range of $\boldsymbol{g}$ has dimension $N = 102$, $g_i(\boldsymbol{y}^d[k]) = \boldsymbol{y}_i^d[k]$ for $i = 1, ..., 27$, and the remaining 75 basis functions $\{g_i : \mathbb{R}^{27} \to \mathbb{R}\}_{i=28}^{102}$ are polynomials of maximum degree 2 that were selected using the SVD dimensional reduction method described in Section 3.1.4.2.

The second Koopman model (with loading) was identified on the same set of snapshot pairs as the first model, but with the loading included $\{\boldsymbol{a}^{(k)}, \boldsymbol{b}^{(k)}, \boldsymbol{u}^{(k)}, \boldsymbol{w}^{(k)}\}_{k=1}^{K}$. The lifting function $\boldsymbol{\psi} : \mathbb{R}^{27\times 9} \to \mathbb{R}^{231}$ was defined as,

$$\boldsymbol{\psi}(\boldsymbol{y}^d[k], \boldsymbol{u}[k], \boldsymbol{w}[k]) = \begin{bmatrix} \gamma(\boldsymbol{y}^d[k], \boldsymbol{w}[k]) \\ \boldsymbol{u}[k] \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{y}^d[k]) \\ \boldsymbol{g}(\boldsymbol{y}^d[k])\boldsymbol{w}[k] \\ \boldsymbol{u}[k] \end{bmatrix} \tag{5.24}$$

where the range of $\boldsymbol{g}$ has dimension $N = 111$, $g_i(\boldsymbol{y}^d[k]) = \boldsymbol{y}_i^d[k]$ for $i = 1, ..., 27$, and the remaining 84 basis functions $\{g_i : \mathbb{R}^n \to \mathbb{R}\}_{i=28}^{111}$ are polynomials of maximum degree 2 that were selected using the SVD dimensional reduction method described in Section 3.1.4.2 once again.

5.3.2 Description of Controllers

Three model predictive controllers were constructed using the data-driven models described in the previous section. Each controller uses one of the identified models to compute online predictions and is denoted by an abbreviation specifying which model,

- L-MPC: Uses the linear state-space model
- K-MPC: Uses the Koopman model without loading
- KL-MPC: Uses the Koopman model with loading

All three controllers solve a quadratic program at each time step using the Gurobi Optimization software [91]. They run in closed-loop at 12 Hz, feature an MPC horizon of 1 second ($N_h = 12$), and a cost function that penalizes deviations of the position of the end effector from a reference trajectory over the prediction horizon.

5.3.3 Experiment 1: Trajectory Following with Known Payload

We first evaluated the relative performance of the three controllers when the payload at the end effector is known. With this information given, the manipulator is tasked with moving the end effector along a three-dimensional reference trajectory lasting 20 seconds. Six trials were completed for payloads of 25, 75, 125, 175, 225, and 275 grams. The actual paths traced out by the end effector and the tracking error over time for 3 of the trials are displayed in Fig. 5.5, and the RMSE tracking error for all 6 trials is compiled in Table 5.2. It should be noted that only the KL-MPC controller is capable of actually utilizing knowledge of the payload, since the other 2 controllers are based on models that do not incorporate loading conditions.

5.3.4 Experiment 2: Online Estimation of Unknown Payload

We evaluated the performance of the online load observer under randomized "ramp and hold" type inputs and a sampling time of $T_s = 0.083$ seconds. New estimates were calculated every $N_e = 12$

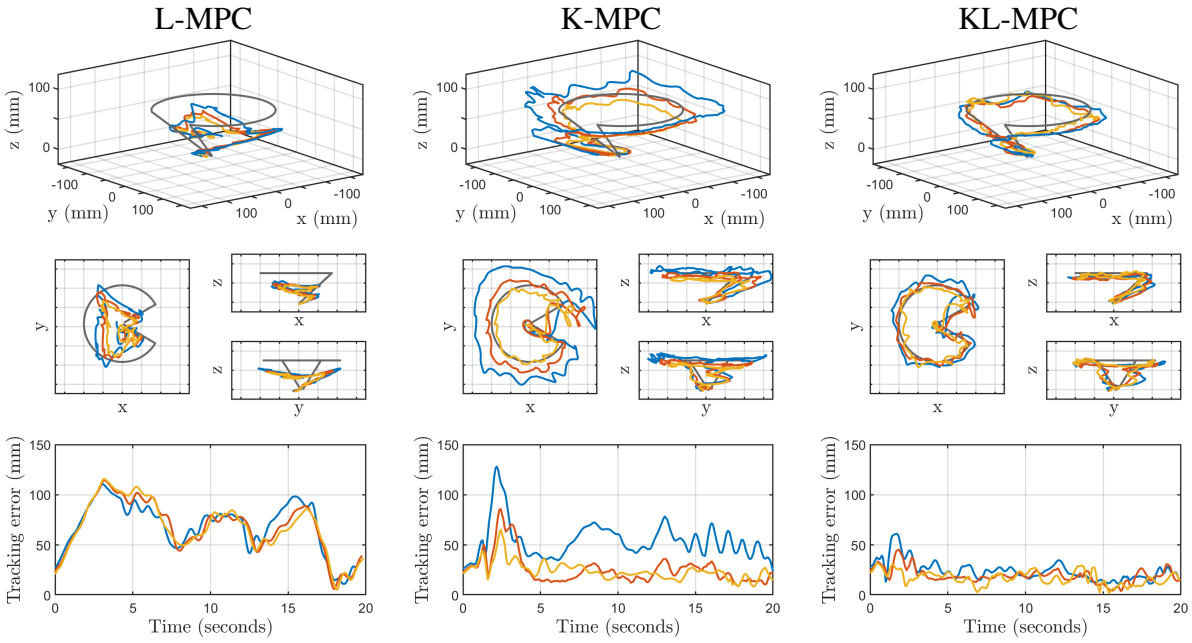


Figure 5.5: Experiment 1 Results: The end effector trajectories for the L-MPC (left), K-MPC (center), and KL-MPC (right) controllers when the true value of the payload is known. Trajectories corresponding to a payload of 25g are shown in blue, trajectories with a payload of 125g are shown in red, trajectories with a payload of 225g are shown in yellow, and the reference trajectory is shown in grey.

Table 5.2: Experiment 1: RMSE (mm) over entire trial

Controller	**Payloads (grams)**						**Avg.**	**Std. Dev.**
	25	75	125	175	225	275		
L-MPC	73.0	72.9	72.6	71.9	72.3	74.3	72.8	0.8
K-MPC	55.4	33.9	29.5	20.0	24.8	27.8	31.9	12.4
KL-MPC	**26.1**	**23.7**	**20.6**	**19.5**	**18.2**	**20.4**	**21.4**	**2.9**

time-steps by solving (5.7) using measurements from the previous $N_w = 30$ time-steps, and $\bar{w}$ was computed by averaging over the most recent $N_r = 360$ estimates. Three trials were conducted with payloads of 25, 125, and 225 grams, none of which were in the set of payloads used for system identification, and the results are displayed in Fig. 5.6

5.3.5 Experiment 3: Trajectory Following with Unknown Payload

To evaluate the efficacy of the combined control and load estimation method summarized by Algorithm 8, we measured the manipulator's performance in tracking a periodic reference trajectory

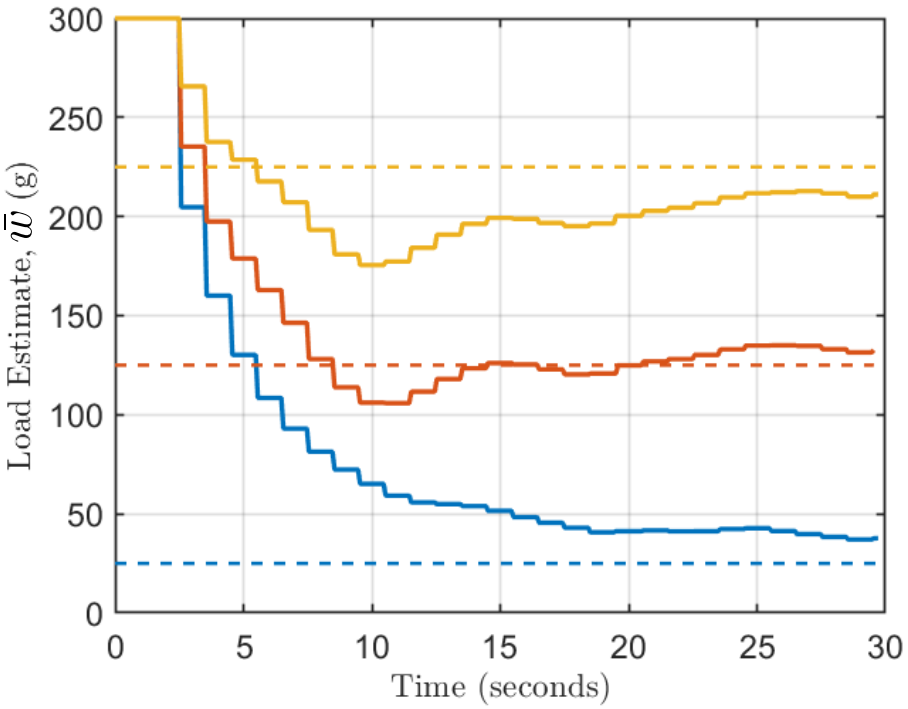


Figure 5.6: Experiment 2 Results: Online payload estimation under random inputs using the method described in Section 5.1.1. Three trials are shown for payloads of 25g, 125g, and 225g, with the actual payload used for each trial marked by a dotted line, and the payload estimate marked a solid line. Results for the 25g payload are shown in blue, results for the 125g payload are shown in red, and results for the 225g payload are shown in yellow.

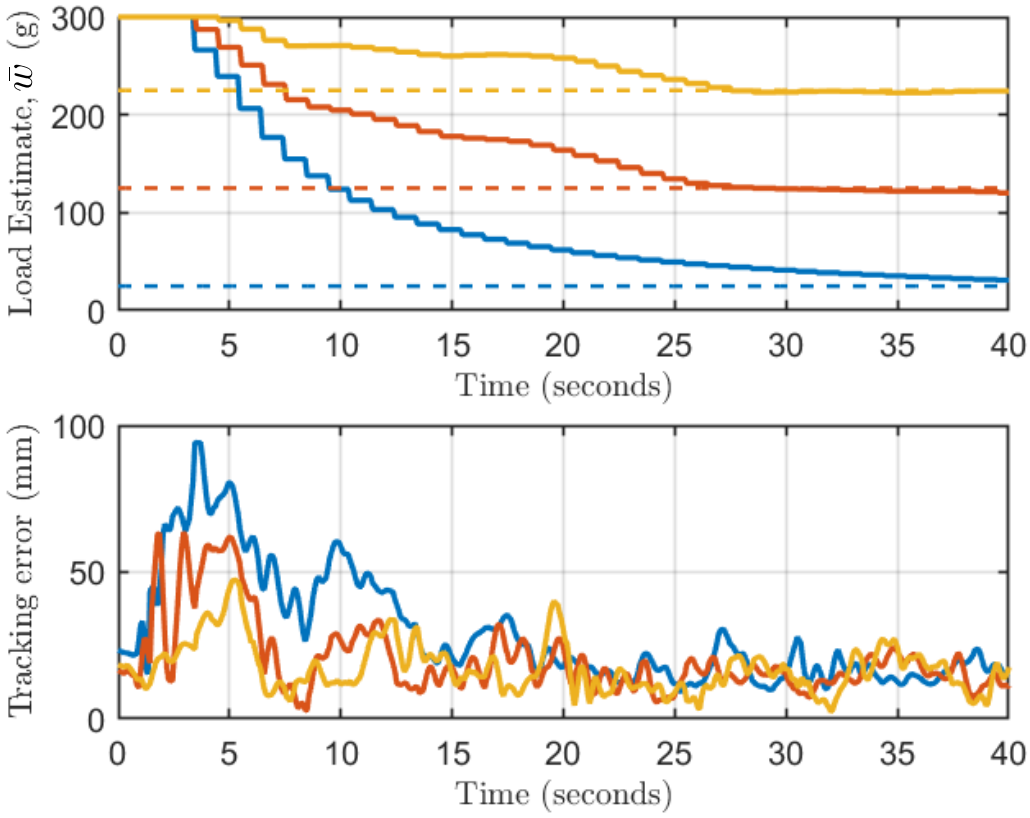


Figure 5.7: Experiment 3 Results: Periodic trajectory following with an unknown payload. The payload estimate over time (top) and tracking error over time (bottom) are shown for three trials with payloads of 25g, 125g, and 225g. Results for the 25g payload trial are shown in blue, results for the 125g payload trial are shown in red, and results for the 225g payload trial are shown in yellow.

when the payload is not known. Once again three trials were conducted with payloads of 25, 125, and 225 grams. The periodic reference trajectory was a circle with a diameter of 200 mm. Note that this trajectory was not part of the training data. The KL-MPC controller was run at 12 Hz and $\bar{w}$ was updated according to the same parameters as in Experiment 2 ($N_e = 12, N_w = 30, N_r = 360$). Results of this experiment are shown in Fig. 5.7.

5.3.6 Experiment 4: Automated Object Sorting (Pick and Place)

The load estimation algorithm and KL-MPC controller were utilized to perform automated object sorting by mass. Five objects were selected, each with mass between 0 and 250 grams, and five

Figure 5.8: Objects used for Experiment 4: In each trial, the soft manipulator sorted a set of five objects according to their mass, based on an estimate computed by the online observer described in Section 5.1.1. The set of objects used for each trial are separated by row, and the mass of each object is written below it.

cups were placed in front of the manipulator, each corresponding to a 50 gram interval between 0 and 250 grams (i.e. 0-50, 50-100, etc.). The range from 250-300 grams was not used for this experiment, because such loads too severely reduce the workspace of the robot. The objects used and their masses are shown in Fig. 5.8. Given one of these objects, the task was to place the object into the cup corresponding to its mass. For each trial, a human assists the manipulator with grabbing the object, then the manipulator performs KL-MPC with load estimation (Algorithm 8) while following a circular reference trajectory for 15 seconds. After 15 seconds, load estimation stops, and a "drop-off" reference trajectory is selected that will move the end effector towards the cup corresponding to the most recent payload estimate. The manipulator then uses KL-MPC to follow the "drop-off" trajectory and deposits the object into the cup. This cycle repeats until all 5 objects are sorted into the proper cup. Using this strategy, the manipulator properly sorted 5 out of 5 objects in 2 separate trials, using a different set of objects each time (see Fig. 5.8). Footage of these trials can be seen in a supplementary video file[2].

[2] https://youtu.be/g2yRUoPK40c

5.3.7 Discussion

This section uses a Koopman operator based approach to model and estimate a variable payload of a soft continuum manipulator arm and employs this knowledge to improve control performance. Experiments confirms that incorporating knowledge about the payload into the model improves tracking accuracy and makes the controller more robust to changes in the loading conditions. In Experiment 1, the KL-MPC controller, which incorporated the payload value, reduced the RMSE tracking error averaged over all payloads by approximately 33% compared to the K-MPC controller that did not utilize information about the payload and reduced the standard deviation of the tracking error by about 77% (see Table 5.2).

To automate the process of identifying payload values, we implemented an observer that was able to automatically estimate unknown payloads within 25 grams in a time of about 15 seconds (see Fig. 5.6). It is notable that this approach was capable of estimating loads other than those presented in the training data set that was used during model-identification. We did not observe over-fitting to the behavior seen under limited loading conditions which suggests that, despite the fact that the approach is data-driven, the identified Koopman model is able to capture the actual physical effect of various loading conditions.

By combining the estimation, modeling, and control into a single MPC algorithm (Algorithm 8), we demonstrated the effectiveness of our approach to improve control accuracy under unknown loading conditions. We first tracked periodic trajectories with an unknown payload. Since the controller needs some time to establish an accurate estimate of the load, the tracking error gradually decreases over time as the load estimate becomes more reliable. After approximately 15 seconds, the tracking error decreased to less than 30 mm, which was about equal to the error with a known load value.

As a final demonstration, we implemented successful pick-and-(mass-based)-place object manipulation using the same algorithm. Unknown objects were successfully sorted by mass, taking advantage of the fact that the payload estimate was accurate enough to choose the correct container for each object and that the tracking error of the "drop-off" trajectory was small enough not

to miss the cup. This required a payload estimate accuracy of less than 50 grams, and a tracking error accuracy of less than 45 mm (the radius of the cups).

While the manipulator exhibited sufficient accuracy to complete this task, several modifications could be made to the robot and controller to improve performance even further. First, the workspace of the manipulator could be greatly enlarged by replacing some of the current actuators with more powerful ones. This could be done without significantly increasing size or weight just by increasing the diameter of the PAMs [48]. Second, a model and controller could be identified with a shorter sampling time, which would enable the model to account for higher frequency behavior and track more dynamic trajectories. This could be achieved by making upgrades to our computational hardware and optimizing our code. Even with these changes, the system's inherent stochasticity would limit tracking accuracy, but these improvements would likely enable much more accurate control.

CHAPTER 6

Conclusion

The goal of this dissertation was to increase the capabilities of soft robots by providing a universal modeling and control framework for them. To that end, two distinct modeling approaches were presented. The first, a static modeling approach for fluid-driven systems, provides a geometry dependent mapping between fluid pressure and force. Because pneumatics and hydraulics remain the most common methods of actuation for soft robotic systems, this framework has broad relevance across the field. The second, a dynamic modeling approach based on Koopman operator theory, provides a data-driven way to represent the input-output behavior of arbitrary soft robotic systems in a computationally efficient manner. These modeling approaches combined with model-based optimal control techniques comprise a novel modeling and control framework catered specifically to soft robots.

The results presented in this dissertation constitute a clear advancement of the state of the art, but there is still much work left to do before the capabilities of soft robots are on par with their rigid-bodied counterparts. This chapter seeks to articulate some of the remaining challenges and avenues for future work in light of the present contributions.

6.1 Discussion of Contributions

The value of the modeling and control framework presented in this dissertation lies in its ability to be applied to any soft robotic system. Hopefully by reducing the time and effort spent developing

custom models for each robot this framework will accelerate the development and deployment of novel soft robotic systems. Each chapter presents one aspect of this overall framework:

Chapter 2 presents a static modeling approach for systems of fluid-driven actuators, which are the most common type of actuators used in soft robots. Unlike the the data-driven approach presented and utilized in later chapters, this approach is physics-based. This means that it can be used to inform the design of hypothetical soft robots before actually constructing them. The downside of this modeling approach is that it is based on a myriad of simplifying assumptions which limit its accuracy. While a valuable design tool, the approach does not yield models that are sufficiently accurate for control.

Chapter 3 presents a dynamic modeling approach for finite-dimensional (nonlinear) dynamical systems. Unlike the approach from Chapter 2, this one is data-driven and avoids physics-based simplifying assumptions. This approach cannot be used to inform design, but captures the behavior of soft robots accurately enough to inform control. Unlike many other data-driven models, which identify black-box models via nonlinear optimization, this approach identifies control-oriented models via linear regression. This makes accurate dynamical models easier to construct, enabling the rapid development of control strategies that exploit the unique characteristics of soft robots.

Chapter 4 builds on the contributions of Chapter 3 by incorporating data-driven models into a model-based optimal control scheme. This scheme was proven capable of achieving accurate real-time control of a real soft robotic system, and outperforming several other benchmark controllers.

Chapter 5 extends the modeling and control methods from Chapter 3 and Chapter 4 to accommodate loading conditions. These loading conditions need not be external forces or masses, but can be a representation of any external disturbances that modify the dynamics of the system. Using this approach, we demonstrated real-time, fully autonomous control of a pneumatically actuated soft continuum manipulator, and executed the first successful demonstration of a soft manipulator performing a pick and place task for objects of unknown mass. While, so far, the approach has only been validated on one specific instance of a pneumatically-actuated soft manipulator, it theoretically should be compatible with other types and classes of soft robotic systems. We thus believe

that the work presented in this dissertation lays the foundation towards enabling the widespread use of automated soft robots in real-world applications.

6.2 Future Directions

The ultimate aim of this work is to actualize soft robots that are capable of deftly assisting humans within in complex, unpredictable environments. This dissertation make progress towards this goal, but falls short of achieving it. This section outlines some of the remaining challenges and potential avenues for future investigation.

6.2.1 Optimal Bilinear Control

In Chapter 3 it was shown that, for some systems, bilinear Koopman realizations are are significantly more accurate than linear realizations and significantly less computationally complex than nonlinear Koopman realizations. Chapter 4 illustrated how this feature could be utilized in a sub-optimal bilinear MPC controller to achieve control accuracy exceeding that of optimal linear and nonlinear MPC controllers. Further work should investigate theoretical guarantees for bilinear controllers, and explore new approaches to optimal control of bilinear dynamical systems.

6.2.2 Model Adaptation Over Time

A downside of the Koopman-based modeling approach presented herein is that models are identified offline. Thus, any changes made to a robot after training are not reflected in its model. Soft materials exhibit viscoelastic behaviors that operate on very long time scales (e.g. days, weeks). These effects will not be captured by training data taken in a single day, and over time the dynamics of the system will drift away from the initially identified model. An algorithm that periodically updates its Koopman operator approximation using more recent data could remedy this problem. Since identifying the Koopman operator is a least-squares regression problem, model

updates could be computed online in a similar fashion to the way the load estimates are computed online in Algorithm 8. Further work might investigate optimal model update strategies.

6.2.3 Contact with the Environment

Chapter 5 extends the scope of the Koopman-based modeling and control approach to account for loading conditions by adding them into the model as states. An implicit assumption of this strategy is that the loading conditions continuously vary the dynamics of the systems. This assumption breaks down in the case of contact with immovable objects in the environment such as walls. In this case, a hybrid dynamical model [94] offers a more appropriate way of representing the discrete transition between contact conditions. A hybrid dynamical system framework could be adopted to model soft robots in contact with the environment. In such a framework, several Koopman realizations would be constructed for different contact conditions, then hybrid optimal control approaches would be utilized to perform tasks that involve contact with the environment.

6.2.4 Combining Physics-based and Data-driven models

The data-driven modeling technique presented in Chapter 3 was shown to be capable of accurately predicting the behavior of real robots. However, a shortcoming of all data-driven models is that their predictions can only be trusted within the range of operating conditions observed in their training data set. Physic-based models, on the other hand, offer predictions that are independent from observations. Such models are more broadly generalizable, but typically less accurate than data-driven models under well observed operating conditions. A combined modeling approach that simultaneously utilizes the accuracy of data-driven models and the generalizability of physics-based models would have many benefits.

One such benefit would be the ability to perform sparse model learning. A robot could begin with a physics-based model only, and use its predictions in a model-based control framework to explore its state space and collect data. As more data is collected, a Koopman-based data-driven model could be constructed online to supplement the physics-based model or capture its error

dynamics. Because the Koopman model would no longer be constructed "from scratch", less data would be sufficient for training.

A combined physics-based and data-driven approach might also aid in the construction of hybrid dynamical models. Changes in the loading and contact conditions of the robot could be inferred based on its deviation from the physics-based model predictions. Then, a data-driven model could be constructed for that specific condition, and stored as one of the modes of a hybrid dynamical model. This type of modeling approach would enable soft robots to learn through contact with the environment, leveraging their softness to do so in an inherently safe way.

6.3 Concluding Remarks

In the past several years the field of soft robotics has generated a lot of excitement. New journals [95] and conferences [96] have been created that are dedicated to the subject, and soft robots have been been popularized in the media as the "robots of the future" [97, 98]. Indeed, it is exciting to imagine a future in which soft robots assist with cooking and cleaning in the home, deploy in disasters to perform search and rescue operations, aid with medical procedures, and venture into outer space to explore new worlds. Despite the optimism surrounding the field, however, the remaining challenges to actualizing this future are numerous and nontrivial. For soft robots to be valued for more than their novelty, they need to prove capable of reliably performing tasks that are actually useful. The tools developed in this dissertation lay the groundwork for making that happen, and will hopefully help lead to the development of highly capable soft robots in the future.

BIBLIOGRAPHY

[1] Amend, J. R., Brown, E., Rodenberg, N., Jaeger, H. M., and Lipson, H., "A positive pressure universal gripper based on the jamming of granular material," *IEEE Transactions on Robotics*, Vol. 28, No. 2, 2012, pp. 341–350. xiii, 113, 114

[2] Graetz, G. and Michaels, G., "Robots at work," *Review of Economics and Statistics*, Vol. 100, No. 5, 2018, pp. 753–768. 1

[3] Dynamics, B., "Spot," 2020. 2

[4] Robotics, F., "Fetch Mobile Manipulator," 2020. 2

[5] News, W., "Security Robot Hurts Child," 2016. 2

[6] Kim, S., Laschi, C., and Trimmer, B., "Soft robotics: a bioinspired evolution in robotics," *Trends in biotechnology*, Vol. 31, No. 5, 2013, pp. 287–294. 2

[7] Rus, D. and Tolley, M. T., "Design, fabrication and control of soft robots," *Nature*, Vol. 521, No. 7553, 2015, pp. 467. 3, 10, 36

[8] Ilievski, F., Mazzeo, A. D., Shepherd, R. F., Chen, X., and Whitesides, G. M., "Soft robotics for chemists," *Angewandte Chemie*, Vol. 123, No. 8, 2011, pp. 1930–1935. 3

[9] Tolley, M. T., Shepherd, R. F., Mosadegh, B., Galloway, K. C., Wehner, M., Karpelson, M., Wood, R. J., and Whitesides, G. M., "A resilient, untethered soft robot," *Soft robotics*, Vol. 1, No. 3, 2014, pp. 213–223. 3, 9

[10] Marchese, A. D., Onal, C. D., and Rus, D., "Autonomous soft robotic fish capable of escape maneuvers using fluidic elastomer actuators," *Soft Robotics*, Vol. 1, No. 1, 2014, pp. 75–87. 3, 9

[11] Miller, D. P., "Assistive robotics: an overview," *Assistive Technology and Artificial Intelligence*, Springer, 1998, pp. 126–136. 3

[12] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R., "Analysis and observations from the first amazon picking challenge," *IEEE Transactions on Automation Science and Engineering*, Vol. 15, No. 1, 2016, pp. 172–188. 3

[13] Bishop-Moser, J. and Kota, S., "Design and modeling of generalized fiber-reinforced pneumatic soft actuators," *IEEE Transactions on Robotics*, Vol. 31, No. 3, 2015, pp. 536–545. 4, 13, 14, 16, 17

[14] Felt, W. and Remy, C. D., "A Closed-Form Kinematic Model for Fiber-Reinforced Elastomeric Enclosures," *Journal of Mechanisms and Robotics*, Vol. 10, No. 1, 2018, pp. 014501. 4, 14

[15] Renda, F., Giorelli, M., Calisti, M., Cianchetti, M., and Laschi, C., "Dynamic model of a multibending soft robot arm driven by cables," *IEEE Transactions on Robotics*, Vol. 30, No. 5, 2014, pp. 1109–1122. 4

[16] Neppalli, S., Csencsits, M. A., Jones, B. A., and Walker, I. D., "Closed-form inverse kinematics for continuum manipulators," *Advanced Robotics*, Vol. 23, No. 15, 2009, pp. 2077–2091. 4

[17] Trivedi, D., Dienno, D., and Rahn, C. D., "Optimal, model-based design of soft robotic manipulators," *Journal of Mechanical Design*, Vol. 130, No. 9, 2008. 4

[18] Webster III, R. J. and Jones, B. A., "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, Vol. 29, No. 13, 2010, pp. 1661–1683. 4, 36, 74

[19] Howell, L. L., Midha, A., and Norton, T., "Evaluation of equivalent spring stiffness for use in a pseudo-rigid-body model of large-deflection compliant mechanisms," *Journal of Mechanical Design*, Vol. 118, No. 1, 1996, pp. 126–131. 4, 36

[20] Katzschmann, R. K., Della Santina, C., Toshimitsu, Y., Bicchi, A., and Rus, D., "Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model," *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, IEEE, 2019, pp. 454–461. 4

[21] George Thuruthel, T., Ansari, Y., Falotico, E., and Laschi, C., "Control Strategies for Soft Robotic Manipulators: A Survey," *Soft robotics*, Vol. 5, No. 2, 2018, pp. 149–163. 4, 74

[22] Gravagne, I. A., Rahn, C. D., and Walker, I. D., "Large deflection dynamics and control for planar continuum robots," *IEEE/ASME transactions on mechatronics*, Vol. 8, No. 2, 2003, pp. 299–307. 4, 36

[23] Trivedi, D., Lotfi, A., and Rahn, C. D., "Geometrically exact models for soft robotic manipulators," *IEEE Transactions on Robotics*, Vol. 24, No. 4, 2008, pp. 773–780. 4, 36

[24] Bruder, D., Sedal, A., Bishop-Moser, J., Kota, S., and Vasudevan, R., "Model based control of fiber reinforced elastofluidic enclosures," *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 5539–5544. 4, 7, 10, 14, 16, 21, 30, 35, 36

[25] Sedal, A., Bruder, D., Bishop-Moser, J., Vasudevan, R., and Kota, S., "A constitutive model for torsional loads on fluid-driven soft robots," *ASME 2017 International Design Engineering*

Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2017, pp. V05AT08A016–V05AT08A016. 4, 14, 21, 35, 36

[26] Bishop-Moser, J., Krishnan, G., Kim, C., and Kota, S., "Design of soft robotic actuators using fluid-filled fiber-reinforced elastomeric enclosures in parallel combinations," *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, pp. 4264–4269. 4, 16, 36

[27] Satheeshbabu, S., Uppalapati, N. K., Chowdhary, G., and Krishnan, G., "Open loop position control of soft continuum arm using deep reinforcement learning," *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 5133–5139. 5, 73

[28] Hyatt, P., Wingate, D., and Killpack, M. D., "Model-based control of soft actuators using learned non-linear discrete-time models," *Front. Robot. AI 6: 22. doi: 10.3389/frobt*, 2019. 5

[29] Thuruthel, T. G., Falotico, E., Renda, F., and Laschi, C., "Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators," *IEEE Transactions on Robotics*, 2018. 5, 36, 73

[30] Boyd, S. and Vandenberghe, L., *Convex optimization*, Cambridge university press, 2004. 5, 36, 75, 76

[31] Bakker, C., Rosenthal, S., and Nowak, K. E., "Koopman Representations of Dynamic Systems with Control," *arXiv preprint arXiv:1908.02233*, 2019. 6, 74

[32] Budišić, M., Mohr, R., and Mezić, I., "Applied koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, Vol. 22, No. 4, 2012, pp. 047510. 6, 38, 40

[33] Bruce, A. L., Zeidan, V. M., and Bernstein, D. S., "What is the koopman operator? a simplified treatment for discrete-time systems," *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 1912–1917. 6

[34] Brunton, S. L., Brunton, B. W., Proctor, J. L., and Kutz, J. N., "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PloS one*, Vol. 11, No. 2, 2016. 6, 74

[35] Williams, M. O., Kevrekidis, I. G., and Rowley, C. W., "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, Vol. 25, No. 6, 2015, pp. 1307–1346. 6, 38, 41

[36] Korda, M. and Mezić, I., "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, Vol. 93, 2018, pp. 149–160. 6, 74, 77

[37] Abraham, I. and Murphey, T. D., "Active Learning of Dynamics for Data-Driven Control Using Koopman Operators," *IEEE Transactions on Robotics*, Vol. 35, No. 5, 2019, pp. 1071–1083. 6

[38] Mamakoukas, G., Castano, M., Tan, X., and Murphey, T., "Local koopman operators for data-driven control of robotic systems," *Robotics: science and systems*, 2019. 6, 74

[39] Bruder, D., Sedal, A., Vasudevan, R., and Remy, C. D., "Force Generation by Parallel Combinations of Fiber-Reinforced Fluid-Driven Actuators," *IEEE Robotics and Automation Letters*, Vol. 3, No. 4, Oct 2018, pp. 3999–4006. 7, 10, 36

[40] Bruder, D., Remy, C. D., and Vasudevan, R., "Nonlinear system identification of soft robot dynamics using koopman operator theory," *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 6244–6250. 7, 39

[41] Bruder, D., Gillespie, B., Remy, C. D., and Vasudevan, R., "Modeling and Control of Soft Robots Using the Koopman Operator and Model Predictive Control," *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. 7, 75

[42] Spong, M. and Vidyasagar, M., *Robot Dynamics And Control*, Wiley India Pvt. Limited, 2008. 9

[43] Grissom, M. D., Chitrakaran, V., Dienno, D., Csencits, M., Pritts, M., Jones, B., McMahan, W., Dawson, D., Rahn, C., and Walker, I., "Design and experimental testing of the octarm soft robot manipulator," *Unmanned Systems Technology VIII*, Vol. 6230, International Society for Optics and Photonics, 2006, p. 62301F. 9, 10, 13

[44] Hawkes, E. W., Blumenschein, L. H., Greer, J. D., and Okamura, A. M., "A soft robot that navigates its environment through growth," *Science Robotics*, Vol. 2, No. 8, 2017, pp. eaan3028. 9

[45] Galloway, K. C., Polygerinos, P., Walsh, C. J., and Wood, R. J., "Mechanically programmable bend radius for fiber-reinforced soft actuators," *Advanced Robotics (ICAR), 2013 16th International Conference on*, IEEE, 2013, pp. 1–6. 10, 13

[46] Marchese, A. D., Katzschmann, R. K., and Rus, D., "A recipe for soft fluidic elastomer robots," *Soft Robotics*, Vol. 2, No. 1, 2015, pp. 7–25. 10

[47] Pridham, J. D. C., "Bellows actuator," May 16 1967, US Patent 3,319,532. 10

[48] Tondu, B., "Modelling of the McKibben artificial muscle: A review," *Journal of Intelligent Material Systems and Structures*, Vol. 23, No. 3, 2012, pp. 225–253. 10, 14, 88, 113, 123

[49] Mosadegh, B., Polygerinos, P., Keplinger, C., Wennstedt, S., Shepherd, R. F., Gupta, U., Shim, J., Bertoldi, K., Walsh, C. J., and Whitesides, G. M., "Pneumatic networks for soft robotics that actuate rapidly," *Advanced functional materials*, Vol. 24, No. 15, 2014, pp. 2163–2170. 10

[50] Hughes, J., Culha, U., Giardina, F., Guenther, F., Rosendo, A., and Iida, F., "Soft manipulators and grippers: a review," *Frontiers in Robotics and AI*, Vol. 3, 2016, pp. 69. 10

[51] Krishnan, G., Bishop-Moser, J., Kim, C., and Kota, S., "Evaluating mobility behavior of fluid filled fiber-reinforced elastomeric enclosures," *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, 2012, pp. 1089–1099. 13, 14

[52] Bishop-Moser, J., Krishnan, G., and Kota, S., "Force and moment generation of fiber-reinforced pneumatic soft actuators," *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, 2013, pp. 4460–4465. 13, 14

[53] Connolly, F., Polygerinos, P., Walsh, C. J., and Bertoldi, K., "Mechanical programming of soft actuators by varying fiber angle," *Soft Robotics*, Vol. 2, No. 1, 2015, pp. 26–32. 13, 14, 21, 35

[54] Connolly, F., Walsh, C. J., and Bertoldi, K., "Automatic design of fiber-reinforced soft actuators for trajectory matching," *Proceedings of the National Academy of Sciences*, Vol. 114, No. 1, 2017, pp. 51–56. 13

[55] Gilbertson, M. D., McDonald, G., Korinek, G., Van de Ven, J. D., and Kowalewski, T. M., "Soft Passive Valves for Serial Actuation in a Soft Hydraulic Robotic Catheter," *Journal of Medical Devices*, Vol. 10, No. 3, 2016, pp. 030931. 13

[56] Krishnan, G., Bishop-Moser, J., Kim, C., and Kota, S., "Kinematics of a generalized class of pneumatic artificial muscles," *Journal of Mechanisms and Robotics*, Vol. 7, No. 4, 2015, pp. 041014. 14

[57] Bishop-Moser, J., Krishnan, G., Kim, C., and Kota, S., "Kinematic synthesis of fiber reinforced soft actuators in parallel combinations," *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, 2012, pp. 1079–1087. 16

[58] Pritts, M. B. and Rahn, C. D., "Design of an artificial muscle continuum robot," *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, Vol. 5, IEEE, 2004, pp. 4742–4746. 16

[59] Sedal, A., Wineman, A., Gillespie, R. B., and Remy, C. D., "Comparison and experimental validation of predictive models for soft, fiber-reinforced actuators," *The International Journal of Robotics Research*, 2019, pp. 0278364919879493. 21, 36

[60] Habibian, S., "Analysis and Control of Fiber-Reinforced Elastomeric Enclosures (FREEs)," *arXiv preprint arXiv:1912.07380*, 2019. 21

[61] Gillespie, M. T., Best, C. M., Townsend, E. C., Wingate, D., and Killpack, M. D., "Learning nonlinear dynamic models of soft robots for model predictive control with neural networks," *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, IEEE, 2018. 36

[62] Ljung, L., *System identification: theory for the user*, Prentice-hall, 1987. 38, 69

[63] Mauroy, A. and Goncalves, J., "Linear identification of nonlinear systems: A lifting technique based on the Koopman operator," *arXiv preprint arXiv:1605.04457*, 2016. 38, 41, 53, 54

[64] Mauroy, A. and Goncalves, J., "Koopman-based lifting techniques for nonlinear systems identification," *arXiv preprint arXiv:1709.02003*, 2017. 38, 41, 54

[65] Penrose, R., "On best approximate solutions of linear matrix equations," *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 52, Cambridge University Press, 1956, pp. 17–19. 42

[66] Lasota, A. and Mackey, M. C., *Chaos, fractals, and noise: stochastic aspects of dynamics*, Vol. 97, Springer Science & Business Media, 2013. 52

[67] Higham, N. J., *Functions of matrices: theory and computation*, Vol. 104, Siam, 2008. 53

[68] Rousseeuw, P. J. and Leroy, A. M., *Robust regression and outlier detection*, Vol. 589, John wiley & sons, 2005. 54

[69] Tibshirani, R., "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996, pp. 267–288. 54, 56

[70] Brunton, S. L. and Kutz, J. N., *Data-driven science and engineering: Machine learning, dynamical systems, and control*, Cambridge University Press, 2019. 59

[71] MATLAB, *version 7.10.0 (R2017a)*, The MathWorks Inc., Natick, Massachusetts, 2017. 69, 91, 92, 115

[72] Felt, W., Lu, S., and Remy, C. D., "Modeling and design of "Smart Braid" inductance sensors for fiber-reinforced elastomeric enclosures," *IEEE Sensors Journal*, Vol. 18, No. 7, 2018, pp. 2827–2835. 74

[73] Felt, W., Telleria, M. J., Allen, T. F., Hein, G., Pompa, J. B., Albert, K., and Remy, C. D., "An inductance-based sensing system for bellows-driven continuum joints in soft robots," *Autonomous Robots*, 2017, pp. 1–14. 74

[74] Yang, S. and Lu, N., "Gauge factor and stretchability of silicon-on-polymer strain gauges," *Sensors*, Vol. 13, No. 7, 2013, pp. 8577–8594. 74

[75] Kim, D.-H., Lu, N., Ma, R., Kim, Y.-S., Kim, R.-H., Wang, S., Wu, J., Won, S. M., Tao, H., Islam, A., et al., "Epidermal electronics," *science*, Vol. 333, No. 6044, 2011, pp. 838–843. 74

[76] Della Santina, C., Bianchi, M., Grioli, G., Angelini, F., Catalano, M., Garabini, M., and Bicchi, A., "Controlling soft robots: balancing feedback and feedforward elements," *IEEE Robotics & Automation Magazine*, Vol. 24, No. 3, 2017, pp. 75–83. 74

[77] Marchese, A. D., Tedrake, R., and Rus, D., "Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator," *The International Journal of Robotics Research*, Vol. 35, No. 8, 2016, pp. 1000–1019. 74

[78] Rawlings, J. B. and Mayne, D. Q., *Model predictive control: Theory and design*, Nob Hill Pub. Madison, Wisconsin, 2009. 74

[79] Paulson, J. A., Mesbah, A., Streif, S., Findeisen, R., and Braatz, R. D., "Fast stochastic model predictive control of high-dimensional systems," *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, IEEE, 2014, pp. 2802–2809. 76

[80] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S., "OSQP: An operator splitting solver for quadratic programs," *2018 UKACC 12th International Conference on Control (CONTROL)*, IEEE, 2018, pp. 339–339. 76

[81] Allgöwer, F. and Zheng, A., *Nonlinear model predictive control*, Vol. 26, Birkhäuser, 2012. 76

[82] Polak, E., *Optimization: algorithms and consistent approximations*, Vol. 124, Springer Science & Business Media, 2012. 76

[83] Patterson, M. A. and Rao, A. V., "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 41, No. 1, 2014, pp. 1. 76, 81

[84] Hereid, A. and Ames, A. D., "FROST: Fast robot optimization and simulation toolkit," *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 719–726. 76, 81

[85] Zhao, P., Mohan, S., and Vasudevan, R., "Control synthesis for nonlinear optimal control via convex relaxations," *American Control Conference (ACC), 2017*, IEEE, 2017, pp. 2654–2661. 76

[86] Aström, K. J. and Murray, R. M., *Feedback systems: an introduction for scientists and engineers*, Princeton university press, 2010. 79

[87] Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M., "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, Vol. 11, No. 1, 2019, pp. 1–36. 81

[88] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, Vol. 47, No. 1, 2005, pp. 99–131. 81

[89] Kalisky, T., Wang, Y., Shih, B., Drotman, D., Jadhav, S., Aronoff-Spencer, E., and Tolley, M. T., "Differential pressure control of 3D printed soft fluidic actuators," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 6207–6213. 88

[90] Van Overschee, P. and De Moor, B., *Subspace identification for linear systems: Theory—Implementation—Applications*, Springer Science & Business Media, 2012. 91, 115

[91] Gurobi Optimization, L., "Gurobi Optimizer Reference Manual," 2018. 97, 117

[92] Colomé, A., Pardo, D., Alenya, G., and Torras, C., "External force estimation during compliant robot manipulation," *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 3535–3540. 101

[93] Zeng, A., Song, S., Lee, J., Rodriquez, A., and A.Funkouser, T., "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics," *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. 101

[94] Schaft, A. and Schumacher, H., *An introduction to hybrid dynamical systems*, 2000. 127

[95] Mary Ann Liebert, Inc., p., "Soft Robotics: Aims and Scope," 2020. 128

[96] Society, I. R. . A., "IEEE International Conference on Soft Robotics," 2020. 128

[97] Shen, H., "Mee the soft, cuddly robots of the future," 2016. 128

[98] Dormehl, L., "Forget metal. When it comes to robots, the future is soft and squishy," 2019. 128