Modeling and Control of Continuum Appendages

by

Xun Fu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in the University of Michigan
2024

Xun Fu

xunfu@umich.edu

ORCID iD: 0000-0001-5000-5190

DEDICATION

In memory of dear family members to whom I could not bid a proper farewell during my PhD journey.

ACKNOWLEDGEMENTS

First, I would like to extend my gratitude to my advisors, Ram Vasudevan and Talia Moore, whose guidance and support have been pivotal in the completion of my PhD.

I am deeply thankful to Ram for teaching me the value of critically examining problems. His thought-provoking questions were stimulating. Now, standing at the conclusion of my PhD journey, I see the immense value of those rigorous discussions. These experiences, although challenging at times, have provided me with invaluable lessons in resilience, intellectual rigor, and problem-solving–lessons that I hope will guide me in all my future endeavors. His ability to navigate complex problems with an abundance of innovative ideas has been nothing short of inspirational.

Equally, I am grateful to Talia, whose unique perspective has enriched this journey immeasurably. Despite coming from a non-engineering background, her openness to learning and ability to introduce fresh, interdisciplinary ideas have been a testament to the value of diverse academic dialogues. Her enthusiasm for sharing knowledge and experiences from various fields has broadened my perspectives. Her adeptness at effective visual communication has generously offered me a new lens through which to view and share research findings, enriching presentations in ways I had not imagined.

I would like to thank the rest of my committee members, Brian and Dan. I highly appreciate Brian's kindness and willingness to answer my research questions, even before I formally asked him to serve on my committee. His insights have been invaluable in refining this dissertation research. I want to give special thanks to Dan for his contributions to the results of this dissertation. Working with Dan and other labmates back in the basement of GG Brown will always be one of my happiest memories at Michigan.

I would like to thank all my colleagues in both the ROAHM lab and the EMBiR lab for making the lab a place of support and collaboration. The constant communication and willingness to help among the labmates have been instrumental in advancing my research progress. The unique talents and diligence each one brings have profoundly inspired me.

Next, I want to thank my friends, those I have had the fortune to meet at Michigan, whose presence has turned my PhD journey into one filled with joy, laughter, and vibrant moments. Their friendship during both the highs and lows of this academic endeavor has been a source of immense comfort and encouragement.

Lastly, I would like to thank my parents and my family for their unwavering love. Their belief in me has been an unspoken promise of support, which has propelled me to explore with confidence and courage.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Appendages such as arms, legs, fins, wings, and tails are peripheral body parts attached to an organism's main body, playing essential roles in animal locomotion. Tails, found in most vertebrates, are particularly versatile, serving a wide range of functions such as providing stability, maneuverability, and prehension. Inspired by these functions, researchers have been integrating tail-like appendages into robotic designs to enhance robot movement, demonstrating significant improvements in control, stability, and efficiency.

However, current modeling studies often simplify animal tails to a single rigid link. While this simplification streamlines the modeling and control of tail-like appendages, it might overlook the potential impacts of having more articulated tails, raising questions about the insights that might be missed by this "reductionist" approach.

Improved models, which incorporate the detailed structure of animal tails, offer a more in-depth approach to uncovering the biological principles of the tail's role in animal locomotion, especially in high performance movements such as rapid aerial reorientation, agile terrestrial maneuvering, and self-righting, which people have a keen interest in. These models can influence the design of bio-inspired robots. Additionally, they hold the potential to guide researchers toward more informative and appropriate simplified models that might not solely consist of a single rigid link. However, while holding considerable value for advancing both biological understanding and robotic research, these improved models exhibit much greater complexity. This renders the analysis and control of them computationally challenging. Hence, devising methods that alleviate the computational load in the analysis and control of complex systems, while still retaining a high degree of accuracy in the depiction of their dynamic behavior, is of paramount importance.

In light of these research gaps, to explore valuable insights into the underlying role of tails in animal movement, this dissertation constructs improved robotic tail models to examine the superior maneuverability afforded by articulated tails over single rigid links and other inertial appendages.

To investigate the often-ignored role of muscle-tendon network actuation in robotic models, we incorporate this network into our analysis by constructing musculoskeletal models of articulated tails. We introduce a specialized software framework for efficiently constructing detailed musculoskeletal models of biological articulated appendages. Using this software and the derived models, we take a nuanced look at the actuation mechanics of biological tails.

To tackle the difficulties faced in analyzing and controlling models of complex systems, such as articulated tails, this dissertation introduces a data-driven modeling and control framework for such systems. This method leverages Koopman operator theory to develop models that are both accurate and computationally efficient, enabling their integration into closed-loop optimal control schemes. The approach shows promise for managing complex biological continuum appendages. It lays the foundation towards achieving real-time control of highly articulated robotic appendages, enhancing the agility of legged robotic systems.

CHAPTER 1

Introduction

1.1 Overview

In the realm of biology, appendages refer to external body parts that are attached to an organism's main body. For example, in humans, arms and legs are examples of appendages. Other examples include the fins of fish, the wings of birds, and the tails of squirrels, all of which play critical roles in animal movement. Among these various appendages, tails, a form of continuum appendage, are widespread across the animal kingdom. Most vertebrates, that is, animals with a spinal column, have evident tails. Certain invertebrates, including scorpions and horseshoe crabs, also possess tail-like appendages.

Despite their common presence in many animal body plans, tails serve extraordinarily diverse functions and are utilized in a remarkably wide range of manners. These functions include, but are not limited to, maneuverability, stability, manipulation, propulsion, defense, energy storage, etc [57]. Geckos, for instance, rely heavily on their tails for crucial support and reactive control during rapid climbing and aerial maneuvers [64]. The remarkable athleticism of cheetahs, especially evident during hunting in the wild, involves significant tail movements for rapid braking, accelerating, and turning [139]. Kangaroos use their tails not only for balance while hopping but also for extra support when sitting upright [97]. Fish propel themselves through water using their tails [33].

Drawing inspiration from these biological examples, numerous researchers have explored integrating tail-inspired appendages into robotic designs to improve robot movement performance

[34, 73, 100, 99, 96, 65, 32, 148, 22]. For instance, Chang et al. [34] and Libby et al. [73], developed a robot equipped with an active tail, allowing for rapid self-righting in the pitch plane. Similarly, Patel et al. [100, 99] demonstrated that a mobile robot with an active tail could turn at higher speeds compared to a counterpart without a tail. Recently, Norby et al. [96] discovered that the aerodynamics of cheetah tails are significantly important in their dynamic motions. By equipping a quadrupedal robot with a tail designed for aerodynamic drag, they demonstrated that such a tail can significantly enhance control while also reducing payload and energy requirements. Rather than solely concentrating on this objective, a considerable group of researchers has been exploring the integration of tails into legged locomotion, where the legs maintain contact with substrates. Yang et al. showed that combining proprioception and tail control can reduce slips and falls and enhance legged robots' ability to traverse extremely uneven terrains [144]. Likewise, Huang et al. illustrated that a robotic arm that is initially introduced for manipulation, when attached to a legged system, can be used as a tail, proving to be an important asset for maintaining stability during locomotion under substantial disturbances [61].

However, upon reviewing the existing literature, it becomes evident that most studies in the field represent animal tails as a single rigid link. While this approach simplifies the modeling and control of the tail and has been instrumental in uncovering some fundamental functions that tails serve in animal movement, it is noteworthy that the tails of actual animals are far more complex than single rigid pendulums. In reality, they generally comprise multiple vertebrae that are interconnected by articulated joints, resulting in highly articulated appendages with multiple degrees of freedom (DOFs). This discrepancy prompts some straightforward but intriguing questions: Are there discernible advantages to having an articulated tail as opposed to a tail represented as a single rigid link? Does the reduction of complex tails to single rigid links lead to an oversight of potential profound insights into tail's underlying role in movement?

Acknowledging the research gap, some researchers have ventured into developing robots with articulated tails [77, 119, 107]. While their work is pioneering, the exploration of whether such articulation is essential remains cursory, especially considering that introducing articulation likely increases the complexity of design and control. Addressing this gap is crucial for advising researchers in the field before proceeding to construct spatial articulated tail robots merely to mimic nature, based on the potentially premature assumption that the existence of articulated tails in highly maneuverable animals inherently suggests an advantage for maneuverability.

To address the questions outlined above, the development of improved tail models becomes essential. This includes models that strive for a relatively high level of accuracy to determine if there are any advantages to having articulated tails as opposed to a single rigid-link tail. Robotic models in which the articulated tails of animals are represented as a set of interconnected rigid bodies could serve as a starting point. These robotic models enable comparisons with the single rigid-link model while prudently replicating the articulation of animal tails.

To further understand the underlying biological principles of tails' contribution to animal movement, it is worth developing models of a higher level of accuracy in representing real animal tails, potentially even including the modeling of soft tissues. These more accurate models facilitate the seamless integration of actual animal motion data for a deeper understanding of how real creatures employ their tails. An effective way of doing this involves building musculoskeletal models [78]. Musculoskeletal models, which combine conceptual, mechanistic, and mathematical models of bones, joints, muscles, tendons, and ligaments, have advanced our fundamental understanding of animal movement. For example, they have enriched our knowledge of sports performance [81, 36], the cause and treatment of movement disorders [128], and dinosaur running [16]. Furthermore, musculoskeletal models can also shed light on robot design. For example, inspired by the presence of bi-articular muscles in human legs, researchers built a bipedal robot with elastic bi-articular actuators [124]. They show that this design can simplify robot control and reduce energy consumption.

Nevertheless, these improved tail models exhibit much greater complexities compared to the single rigid link, due to the inherent composition of tails, which feature a large number of joints. Furthermore, introducing soft tissues can introduce even more complexities. Such complexities render the analysis and control of these models challenging. Therefore, the ability to accurately capture the dynamics of complex systems while making them more accessible for analysis and control becomes exceedingly valuable.

The goal of this document is to construct comprehensive tail models for gaining valuable insights into the tail's underlying role in animal movement and to formulate methods for creating models that effectively capture the dynamics of complex systems while rendering them more amenable for analysis and control, alongside the development of synthesized controllers.

1.2 Contributions

The first contribution in this dissertation focuses on the comparison of diverse robotic inertial appendages, including a single rigid-link tail, articulated tails, and artificial mechanisms. This study evaluates their effectiveness in facilitating three-dimensional body rotation, which serves as a criterion to measure the maneuvering capabilities of the inertial appendage, through the use of physics-based trajectory optimizations. Our findings indicate that articulated tails, characterized by multiple links, provide enhanced control and superior performance relative to other inertial appendages. Additionally, by optimizing the relative sizes of links within a jointed tail, we observed a pattern in the link length distribution of the optimal configuration. The pattern surprisingly aligns with the those found in the tails of animals known for their high maneuverability, likely suggesting that such a configuration might contribute to the exceptional maneuverability. The findings collectively indicate that a certain degree of tail articulation is important for maneuverability, and it is worthwhile to investigate biological jointed tails in depth to uncover the underlying mechanisms.

The second contribution involves the development of a specialized software framework for musculoskeletal modeling. This software is designed to aid biomechanics researchers in efficiently constructing models of articulated limbs using data from CT scans and dissections. It integrates seamlessly with a well-known open-source software for modeling, simulating, and analyzing musculoskeletal systems, thereby streamlining the model development process and enhancing sharing capabilities within the research community. A distinctive feature of this software is its innovative approach to modeling the mechanics of branched muscle-tendon units. The novel modeling approach explicitly considers the physical interactions between the muscles and tendons within branched architectures, an aspect largely neglected in existing research. To understand the importance of explicitly modeling interactions between elements of branched muscle-tendon units, we conducted a comparative analysis between our proposed branching method and the parallel method commonly used in previous studies with a variety of muscle-tendon architectures. The results revealed large disparities between the two approaches, underscoring the potential critical role of physical interactions within a network of branched tendons on musculoskeletal function.

The third contribution is a linear Koopman realization for data-driven modeling and control of complex systems. The framework is based on the Koopman operator theory. The Koopman-based modeling method allows to effectively represent the dynamics of complex systems as linear dynamical systems by using least-squares linear regression. This approach results in model predictive control optimization problems that are convex, thereby enhancing computational efficiency to levels suitable for real-time closed-loop control. This modeling and control framework is validated on a continuum robotic system, demonstrating its promising potential for extension to intricate systems, including biological continuum appendages.

The fourth contribution is a bilinear Koopman realization for data-driven modeling and control of complex systems. This bilinear approach navigates the limitations encountered with the previously introduced linear Koopman realization, which is there is no theoretical guarantee for the existence of a linear Koopman representation if the approximation is limited to a subspace that excludes nonlinear functions of the input. We provide theoretical proofs demonstrating that Koopman representations incorporating bilinear control input terms are more likely to exist for arbitrary dynamical systems than linear counterparts. By employing this approach, we show that a bilinear realization strikes a balance between the computational efficiency of linear systems and the expressiveness of fully nonlinear systems. This makes them an attractive option for representing systems for which linear models are insufficient but closed-loop controllability is still desired.

1.3 Organization

Each subsequent chapter corresponds to one of the contributions outlined above and shares a similar structure. Each chapter begins with an introduction featuring an elaborate literature review within the context of the corresponding project, followed by an explanation of the methods employed, and then one or more sections describing the results and providing conclusive remarks.

CHAPTER 2

Jointed Tails Enhance Control of Three-dimensional Body Rotation

2.1 Introduction

As exemplified by geckos and agamid lizards, tails can be swung to generate body rotations during aerial maneuvers to rapidly self right and transition from horizontal to vertical movement [64, 73]. This fundamental concept of inertial maneuvering generalizes to any appendage with substantial moment of inertia that can be rotated independently, thereby generating torque and rotation in the body. For example, the substantial mass of bat wings facilitates rapid inertial maneuvering that facilitates inverted perching and recovery from perturbation during flight [20]. Praying mantises exchange the angular momentum between their legs and abdomen to control body pitch when leaping towards a target [29]. Hawkmoths also use their massive abdomen as an inertial appendage in concert with the torques generated by their wings during flight to closely track moving targets [72]. Using physics-based models of inertial maneuvering , we can even estimate the extent to which extinct animals could have used inertial appendages to control body rotations [73].

Drawing inspiration from these biological examples, roboticists have designed inertial appendages to facilitate body rotations in both aerial and terrestrial settings. A straightforward way to generate high torques is with a rotating mass, or reaction wheel, which is commonly used to enhance stability [105]. Unlike tails, reaction wheels are unrestricted in stroke, but require more motion for the same moment of inertia to generate the same torque as a tail [22]. Thus, active tails are move effective than reaction wheels when power and time are limited, such as during rapid maneuvers in both aerial and terrestrial contexts. For instance, an active tail that controls the aerial pitch rotations of a robot ensured safe landings after driving off a ramp [34, 73]. A insect-scale robot used an inertial tail to reorient during flight [126]. Similar studies demonstrated that a mobile robot with an active tail could turn at higher speeds compared to a tailless counterpart [100, 99, 130, 109]. These planar rigid tails have served as physical models to test biological hypotheses, enhance robot maneuverability, and are simple to design and control.

However, the biological inertial appendages are far more diverse and complex than a simple rigid pendulum and can change shape to dynamically vary moment of inertia through time. This discrepancy prompts two major questions: How can we compare the maximum potential effectiveness of diverse inertial appendages that dynamically vary in moment of inertia? Does increasing the number of joints in a tail enhance its function as an inertial appendage? In this paper, we address these two questions by developing physics-based simulations and applying trajectory optimization techniques to compare a variety of inertial appendages.

We explore two major configurations of inertial appendages as they move in 3D space and reconfigure to produce dynamic changes in moment of inertia. The first example is mounting a reaction wheel perpendicularly to the end of a rigid tail. This underactuated configuration, which we call an “extended reaction wheel,” has the benefit of simplifying control inputs. In a previous study, a one degree-of-freedom (DOF) tail moving in a single plane combined with a 1-DOF reaction wheel was capable of generating rotations in all three planes [37]. Here, we use a 2-DOF tail with a 1-DOF reaction wheel, which enables body rotational across all three rotational DOFs independently, likely enhancing the effectiveness of control. We compare this to tails consisting of rigid 2-DOF pendulums, which can also theoretically generate 3D rotations. Although the extended reaction wheel is not found in nature, it is useful for design engineers to compare the effectiveness of each style of inertial appendage.

We also explore the effect of dynamic changes in MOI in the context of jointed tails. For example, the jerboa has the longest tail (relative to body size) for any mammal, and the many joints in the tail allow jerboas to change the shape of their tail as they swing it in 3D space. In particular, we observed that in mammals with the same tail lengths, the number of vertebrae that make up the tail differs between species. Increasing the number of rigid links likely affects the capability of a tail to dynamically change MOI and the complexity of control required to perform those changes. A previous study found that adding joints enhances the inertial effectiveness of a tail in 2D [116], but the prescribed tail trajectory used in the comparison may not maximize torque for all configurations, especially because all joints were programmed with identical rotation. Here, we investigate how increasing the number of rigid links and independently actuated 2-DOF joints within a tail affects its functional capabilities as an inertial appendage in 3D space.

If the objective is to compare the capabilities of these diverse inertial appendages,it is necessary to formulate an optimization problem that maximizes performance under certain constraints. This would ensure that we are fully harnessing the potential of each configuration for comparison. Among the most important constraints to consider are the control input limits. Both in animals and in robotic systems, there are practical limits to the speed of actuation and feedback as well as the complexity of control signals. If tails with different degrees of freedom are subject to different levels of control input limits, it might erroneously suggest that tails with more links are inherently more efficient. Therefore, a fair assessment requires that all tail configurations operate within the same control input parameters, ensuring that any observed differences in performance are truly due to the design variations, not unequal control capabilities.

This study introduces an optimization-based approach evaluate the maximum performance of inertial appendages under the same set of constraints. This method finds the trajectory that minimizes the error to a target set of body rotations for each unique appendage configuration. We found that, under these conditions, a jointed tail comprised of three or more 2DOF rigid link pendulums outperforms an equitable extended reaction wheel tail. Within rigid link tails, we found that when comprised of equally sized, independently actuated links—with the same total tail length and moment of inertia in the fully extended state—increasing the number of links increases performance of the inertial appendage. By allowing the relative size of links within a tail to vary, we sought to further optimize the design of tails as an inertial appendage. We found that the optimal configuration consists of short links in the proximal and distal portions, with the longest links in the mid-region of the tail. Surprisingly, our results match the pattern of vertebral lengths found in mouse tails [125]. This optimization-based simulation is a generalized approach to estimate the function of any inertial appendage, regardless of shape. This can be applied to estimate the potential inertial maneuvering ability of vertebrate tails from simple series of skeletal measurements, compare inertial appendages that differ greatly in morphology and complexity, and provide essential guidance for the design of novel robotic inertial appendages.

2.2 Methods

Here, we isolate the effect of inertial appendages by simulating a 3D environment without gravity, aerodynamics, or contact with a substrate. Based on that, we develop robotic models composed of two components: a torso and an inertial appendage. In this context, the torso rotations we measure can only be induced by movement of the inertial appendage.

We measure the maximal performance of an inertial appendage by using its actuation to make the torso follow a desired "trajectory" of torso orientations. The desired torso trajectories involve large accelerations within a short time, resulting in pitch, roll, and yaw rotations similar to those animals experience when banking for a turn or preparing for a leap. Performance of the inertial appendage is then quantified by assessing how well the torso tracks the desired trajectories through the actuation of the tail under certain constraints. The task is formulated as a trajectory optimization problem. By solving the problem, we can evaluate the performance of appendages with different configurations. Note that the torso is not translating through space in the simulations, simply rotating.

The remainder of this section is organized as follows: We first introduce the robotic model design and associated parameters. Then, we explain how we generate the desired torso orientation trajectories. Following that, we present the formulated trajectory optimization problems.

2.2.1 Model Design and Parameters

In our robotic model, the torso and the inertial appendage are conceptualized as interconnected rigid bodies (see Fig 2.1A). The connection between these components is established through rotational joints. Specifically, the torso is configured as a floating body represented by a uniformly dense square prism, attached to the ground through an unactuated 3-DOF rotational joint, enabling roll, pitch, and yaw movements.

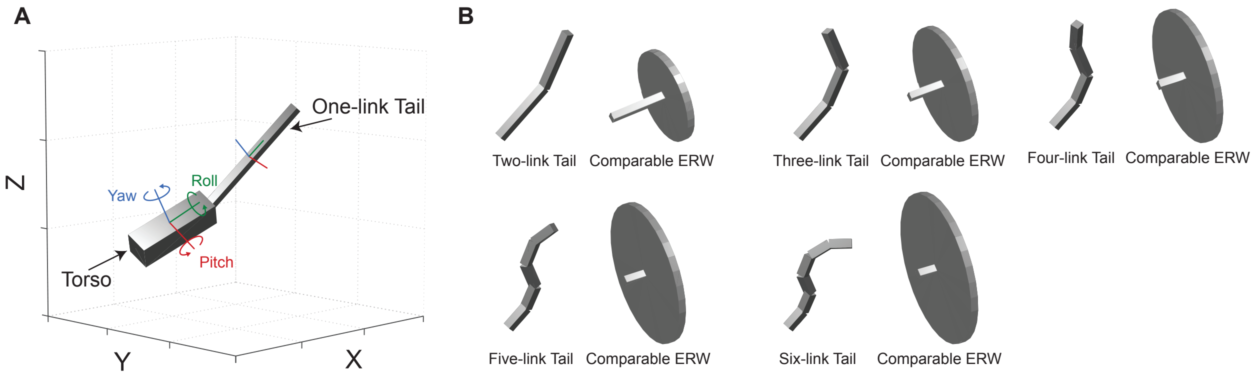

Figure 2.1: **Model visualization.** (**A**) provides an example of the model structure, featuring a torso and a single-link tail. (**B**) demonstrates jointed tails with varying numbers of links (two to six), each with uniform link segment lengths, alongside their corresponding comparable extended reaction wheels. The first row displays jointed tails with two, three, and four links, and their respective extended reaction wheels. The second row presents jointed tails with five and six links, and their corresponding extended reaction wheels. Rotational joints in the models are positioned at non-zero angles to facilitate clear distinction.

In this study, we compare three categories of inertial appendage: a single rigid pendulum (see Fig2.1A), a jointed tail (see Fig2.1B), and an extended reaction wheel (ERW) (see Fig2.1B). The single rigid pendulum is represented by a uniformly dense square prism, and connected to the torso through a 2-DOF joint, which allow for pitch and yaw movements.

The consecutive rigid bodies or links, also represented by uniformly dense square prisms, in each jointed tail are connected to each other and to the torso through the same type of 2-DOF joints. Each DOF of these 2-DOF joints is actuated by an ideal, massless motor, supplying torques for active, independent pitch and yaw movements. Regardless of the number of links, the total mass and length of the jointed tails are all the same. This design choice, particularly the exclusion of a direct roll DOF in the appendage joints, is based on three considerations: the naturally limited range of roll motion in the tails of real animals; the effectiveness of the combination of pitch and yaw movements in manipulating the torso's roll angle without a dedicated roll movement mechanism; and the benefit of reducing the computational load by simplifying the model's joint complexity.

The ERW models are characterized by appendages consisting of a single rigid link directly attached to the torso via the same 2-DOF joint used in the tails of tail models, as outlined above, and a reaction wheel connected to this link via a 1-DOF (roll) joint. Unlike conventional reaction wheels and the configuration described in [37], the link between the wheel and the torso can rotate relative to the torso along both pitch and yaw axes. This capability allows the ERW to reorient the torso across three rotational axes independently.

To aim for an equitable comparison between ERW and jointed tails, we develop five configurations for the ERW (see Fig 2.1B), corresponding jointed tails with two-link tail, three-link tail, four-link tail, five-link tail, and six-link tail, respectively. The mass and dimensions of the link in the ERW are identical to those of the first link in the corresponding jointed tail. The mass of the reaction wheel is set to be equal to the combined mass of the tail in the corresponding tail model, excluding the first caudal link. The reaction wheel's radius is chosen to match the jointed tail moment of inertia around its rotational axis, considering the entire tail beyond the first caudal link rotating around the pitch or yaw axis.

Detailed specifications of the model, including mass, size, joint torques, and other relevant parameters used for this study, are presented in Table 2.1.

Ideally, these parameters would be sourced from empirical data pertaining to real animals, particularly those known for their exceptional use of tail in achieving high maneuverability. For instance, the mass and length of an animal's torso and tail, if available, could be directly integrated into the design of our robotic model. Further, a detailed analysis of the primary muscle groups responsible for tail movement could inform the estimation of maximum joint torques for the robotic tails. This method could extend to other parameters, such as the range of motion and maximum

Table 2.1: Model parameters

Property	**Value**
Torso Mass	5 kg
Appendage Total Mass	1.5 kg
Torso Dimension (width×length×height)	0.3 m × 1 m × 0.3 m
Tail Dimension (width×length×height)	0.1 m × 1.5 m × 0.1 m
Tail Joint Velocity Range	$[-360,\ 360]$ deg/s
Tail Joint Range of Motion	$[-60,\ 60]$ deg
Tail Joint Torque Range	$[-5,\ 5]$ Nm
Wheel Joint Velocity Range	$[-360,\ 360]$ deg/s
Wheel Joint Range of Motion	$[-\infty,\ +\infty]$ deg
Wheel Joint Torque Range	$[-5,\ 5]$ Nm

The numbers in the table for dimensions follow the order of the local X, Y, and Z axes of the coordinate system illustrated in Fig. 2.1, represented by red, green, and blue, respectively

speed of the joints in the model.

Unfortunately, due to the scarcity of comprehensive, specific data in current literature, we opt to select these parameters manually. It's important to note that the chosen parameters, while not replicating the exact characteristics of any particular animal, can facilitate meaningful insights that are broadly applicable and generalizable. Any trends and patterns identified in this study can contribute to a broader understanding of the principles of appendage's contribution to body rotation.

In this study, we present results for tails comprising up to a maximum of six links with uniform tail link segment lengths, due to the escalating computational demands associated with tails with more link segments. As mentioned earlier, we also assess the impact of variations in the tail link segment lengths, where we optimize the tail segment length under the constraint of fixed total tail length. Importantly, in our study, variations in tail link segment lengths are made intrinsically linked with corresponding changes in mass and moment of inertia, providing a more holistic understanding of the implications of size variations. For models with non-uniform link lengths in their tails, we present results for up to four links. The results for the five-link and six-link configurations are not available because we have to employ symbolic representation to capture the dynamics of the robots with non-uniform tail link segment lengths. As the system dimension grows, the process of exporting these symbolic representations for optimization use becomes considerably challenging.

2.2.2 Torso Trajectory Generation

We employ Fourier series to generate the torso's roll, pitch, and yaw rotational trajectories similar to those animals would experience during turning, self-righting, or perturbation recovery. A set of 20 trajectories for pitch, roll, and yaw (see Fig 2.2), each with a duration of 0.5 seconds are generated. To accommodate the physical limitations, the roll, pitch, and yaw angles are constrained within the range of $[-180, 180]$ deg and angular velocities are limited to a range of $[-360, 360]$ deg/s for all 20 torso orientation trajectories.

Here, let $t \in [t_0, t_f]$ represent time. We denote the torso's orientation (i.e., roll, pitch, and yaw angles) by $\Theta : [t_0, t_f] \to \mathbb{R}^3$,

$$\Theta(t) = \Big[\theta_1(t),\ \theta_2(t),\ \theta_3(t)\Big]^\top \tag{2.1}$$

We generate random trajectories for the torso's orientation using a fifth-degree Fourier series as follows:

$$\theta_i(t) = a_{i0} + \sum_{j=1}^{5} (a_{ij} \cos(j\omega_i t) + b_{ij} \sin(j\omega_i t)) \tag{2.2}$$

where $\{a_{ij}\}_{j=0}^5$, $\{b_{ij}\}_{j=1}^5$, and ω_i, for each $i = 1, 2, 3$, are adjustable parameters. Different combinations of pitch, roll, and yaw parameters are used to generate each of the 20 trajectories.

2.2.3 Trajectory Optimization

In this subsection, we describe the formulation of three trajectory optimization problems. The first is for tail models where the length of each individual link in the tail is uniform across all links. The second is for the ERW. The third is for jointed tail models where the length of each individual tail link can vary, while the overall tail length remains fixed.

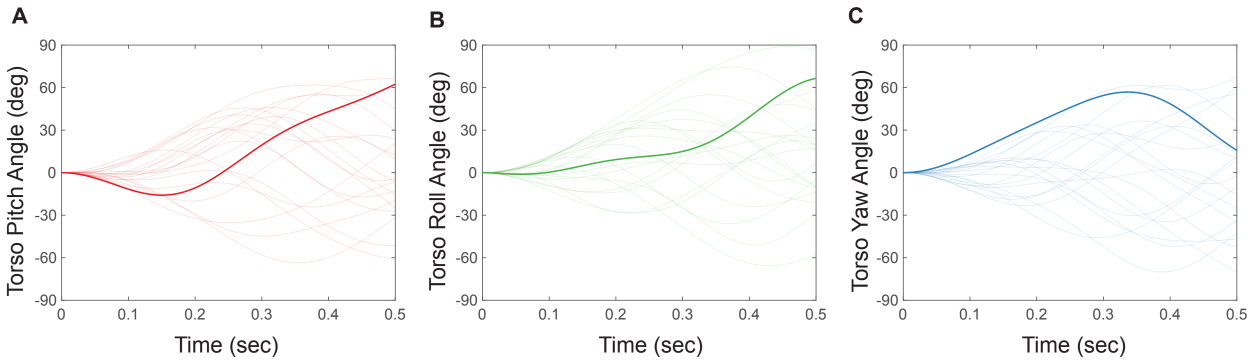

Figure 2.2: **Illustration of designed torso orientation trajectories.** (**A**), (**B**), and (**C**) depict the 20 desired trajectories for torso pitch, roll, and yaw angles, respectively. Shaded lines represent individual trajectories, while solid lines highlight exemplary trajectories to illustrate variations in angles over time. As seen in the figure, within a time span of 0.5 seconds, the orientations undergo rapid changes, experiencing shifts of more than 60 degrees from the starting point. In addition, it can become even more dynamic, with 60-degree changes occurring midway through the 0.5-second interval, followed by a reversal of these changes in the latter half.

2.2.3.1 Uniform Tail Link Length

Given a tail that consists of n_l links, we define the joint configuration trajectory as $q : [t_0, t_f] \to \mathbb{R}^{n_q}$, where $n_q = 2n_l + 3$. For the models in our study, the joint configuration comprises the position of each revolute joint in the tail, with the first three elements corresponding to the torso orientation angles. The joint velocity and joint acceleration trajectories are denoted by $\dot{q} : [t_0, t_f] \to \mathbb{R}^{n_q}$, $\ddot{q} : [t_0, t_f] \to \mathbb{R}^{n_q}$, respectively.

The dynamics of the given model are governed by the following equation:

$$\ddot{q}(t) = -M(q(t))^{-1}\Big(- H(q(t), \dot{q}(t)) + \tau(t)\Big) \tag{2.3}$$

where $\tau(t) \in \mathbb{R}^{n_q}$ represents the joint control inputs, $M(q(t)) \in \mathbb{R}^{n_q \times n_q}$ is the mass-inertia matrix, and $H(q(t), \dot{q}(t)) \in \mathbb{R}^{n_q \times n_q}$ represents the combined effects of Coriolis forces, centrifugal forces, and gravitational forces, all at time t.

Note that, by default, this dynamics equation assumes that all the joints, including the floating joints of the torso, can supply joint torques independently. To emphasize that the task involves only actuating the joints in the appendages to induce torso rotations, we forcibly apply zero torque to the floating joints of the torso:

$$\ddot{q}(t) = M(q(t))^{-1}\Big(- H(q(t), \dot{q}(t)) + \Big[O_{1\text{x}3}, \quad u(t)^\top\Big]^\top \Big) \tag{2.4}$$

where $u(t) \in \mathbb{R}^{n_q - 3}$ represents the torque inputs for the tail joints.

To express the dynamics of the model in a first-order form, as required by standard trajectory optimization, we define the system state trajectory and system dynamics as:

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{2.5}$$

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} \tag{2.6}$$

We define the objective function as follows,

$$\min_{x,u,\dot{u}} \int_{t_0}^{t_f} \| \, Cx(t) - \Theta(t) \, \|_2^2 \, dt \tag{2.7}$$

where $C \in \mathbb{R}^{3 \times 2n_q}$ is a selection matrix used to extract the torso orientations from the system states, and Θ represents the desired torso orientation trajectory. Thus, the cost function measures the integrated tracking error between the actual torso orientations and the desired torso orientations.

Beyond the dynamics outlined earlier, the optimization is subject to other constraints, including the joint position and joint velocity constraints:

$$S_{lb} \leq x(t) \leq S_{ub} \qquad \forall t \in [t_0, t_f] \tag{2.8}$$

where the values of the bounds can be found in Table 2.1, and the system control input constraints:

$$U_{lb} \leq u(t) \leq U_{ub} \qquad \forall t \in [t_0, t_f] \tag{2.9}$$

In addition, to avoid generating solutions with impractical, erratic control inputs, we introduce constraints on the rates of change of the inputs:

$$R_{lb} \leq \dot{u}(t) \leq R_{ub} \qquad \forall t \in [t_0, t_f] \tag{2.10}$$

Importantly, we also include a constraint that limits the control efforts of the tail joints:

$$u(t)^\top u(t) \leq E \qquad \forall t \in [t_0, t_f] \tag{2.11}$$

This constraint is applied to all the models designed in **??**, and the value of the bound E is chosen such that it does not exceed the maximum possible value for the model with a single-link tail. This ensures that the models with different numbers of tail links have the same level of total control effort limit, facilitating an equitable comparison.

Lastly, we include constraints to prevent self-collisions:

$$g(x(t)) \leq O_{n_g \times 1} \qquad \forall t \in [t_0, t_f] \tag{2.12}$$

To prevent the tail from intersecting with the torso, we over-approximate the torso using spheres. Similarly, the joint connections in the tail, as well as the tip of the tail, are approximated by spheres. To avoid contact between the tail and the torso, we formulate constraints such that the distance between each sphere on the tail and each sphere on the torso is greater than the sum of their respective radii. Each element in the function $g : [t_0, t_f] \to \mathbb{R}^{n_g}$ specifies one such relationship between a pair of spheres. The dimension of this constraint, n_g, varies with the number of tail links. It is important to note that, given the range of motion values and the lengths of the tail links, there is no need to explicitly consider self-collision among the tail links. This simplification reduces the number of nonlinear constraints in the optimization problem, thereby reducing the computational load required to solve the problem.

Combining everything together, the complete formulation of the trajectory optimization problem is presented as follows:

$$
\begin{aligned}
\min_{x,u,\dot{u}} \quad & \int_{t_0}^{t_f} \| Cx(t) - \Theta(t) \|_2^2 \, dt & \\
\text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t)) & \forall t \in [t_0, t_f] \\
& S_{lb} \leq x(t) \leq S_{ub} & \forall t \in [t_0, t_f] \\
& U_{lb} \leq u(t) \leq U_{ub} & \forall t \in [t_0, t_f] \\
& R_{lb} \leq \dot{u}(t) \leq R_{ub} & \forall t \in [t_0, t_f] \\
& u(t)^\top u(t) \leq E & \forall t \in [t_0, t_f] \\
& g(x(t)) \leq O_{n_g \times 1} & \forall t \in [t_0, t_f]
\end{aligned}
\tag{2.13}
$$

Methods for solving trajectory optimization problems can be divided into two categories: indirect and direct [15]. Here, we use the direct collocation method, which discretizes the continuous-time trajectory optimization problem by approximating the continuous functions in the problem statement as polynomial splines, thereby converting it into a nonlinear problem. To provide a solution that achieves high-order accuracy without necessitating super fine intervals or a small timestep, which typically increases the computational load, we employ the Hermite-Simpson collocation method. This high-order method approximates continuous functions as piecewise quadratic functions, offering more accurate approximations.

The original trajectory optimization is transcribed as the following nonlinear program:

$$
\begin{aligned}
\min_{\substack{x_0, x_{0+\frac{1}{2}}, \dots, x_N \\ u_0, u_{0+\frac{1}{2}}, \dots, u_N}} \quad & \sum_{k=0}^{N-1} \frac{dt}{6} \Big(\| Cx_k - \Theta_k \|_2^2 + 4 \left\| Cx_{k+\frac{1}{2}} - \Theta_{k+\frac{1}{2}} \right\|_2^2 \\
& + \| Cx_{k+1} - \Theta_{k+1} \|_2^2 \Big) \\
\text{s.t.} \quad & x_{k+1} = x_k + \frac{dt}{6}(f_k + 4f_{k+\frac{1}{2}} + f_{k+1}) && k \in 0, \dots, N-1 \\
& x_{k+\frac{1}{2}} = \frac{1}{2}(x_k + x_{k+1}) + \frac{dt}{8}(f_k - f_{k+1}) && k \in 0, \dots, N-1 \\
& S_{lb} \leq x_k \leq S_{ub}\,, \quad S_{lb} \leq x_{k+\frac{1}{2}} \leq S_{ub} && k \in 0, \dots, N \\
& U_{lb} \leq u_k \leq U_{ub}\,, \quad U_{lb} \leq u_{k+\frac{1}{2}} \leq U_{ub} && k \in 0, \dots, N \\
& U_{lb} \leq u_k \leq U_{ub}\,, \quad U_{lb} \leq u_{k+\frac{1}{2}} \leq U_{ub} && k \in 0, \dots, N \\
& U_{lb} \leq u_k \leq U_{ub}\,, \quad U_{lb} \leq u_{k+\frac{1}{2}} \leq U_{ub} && k \in 0, \dots, N \\
& R_{lb} \leq \frac{u_{k+1} - u_k}{dt} \leq R_{ub} && k \in 0, \dots, N-1 \\
& g_k \leq O_{n_g \times 1}\,, \quad g_{k+\frac{1}{2}} \leq O_{n_g \times 1} && k \in 0, \dots, N
\end{aligned}
\tag{2.14}
$$

where N represents the number of spline segments, t_k represents the time at k-th knot (collocation) point, $dt = t_{k+1} - t_k$ represents the duration of k-th spline segment, $x_k = x(t_k)$, $\Theta_k = \Theta(t_k)$, $u_k = u(t_k)$, $f_k = f(x(t_k), u(t_k))$, $g_k = g(x(t_k))$. Detailed tutorials on transcribing such a trajectory optimization problem to a nonlinear program can be found in [15, 67].

2.2.3.2 Extended Reaction Wheel

Compared to the formulation of the trajectory optimization problem for tail models (2.13), where each tail link length is uniform, the trajectory optimization problem for ERW models exhibits two primary differences. First, the number of joint configurations becomes constant for all ERW systems, that is, $n_q = 6$, with the first three corresponding to the torso orientation—identical to that of tail models—while the last three correspond to two joint positions for the joint connecting the link and the torso, and one joint position for the joint connecting the link and the wheel. Second, due to the model's design, including the joint range of motion and the dimensions of the link and wheel, there is no need to introduce constraints to prevent self-collisions. Consequently, the self-collision constraints (2.12) are removed.

With these modifications, the formulation of the trajectory optimization for ERW models is as follows

$$\begin{aligned}
\min_{x,u,\dot{u}} \quad & \int_{t_0}^{t_f} \| Cx(t) - \Theta(t) \|_2^2 \, dt & \\
\text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t)) & \forall t \in [t_0, t_f] \\
& S_{lb} \leq x(t) \leq S_{ub} & \forall t \in [t_0, t_f] \\
& U_{lb} \leq u(t) \leq U_{ub} & \forall t \in [t_0, t_f] \\
& R_{lb} \leq \dot{u}(t) \leq R_{ub} & \forall t \in [t_0, t_f] \\
& u(t)^\top u(t) \leq E & \forall t \in [t_0, t_f]
\end{aligned} \tag{2.15}$$

To solve this, we convert it into a nonlinear program by following the same process as previously presented. For succinctness, the detailed nonlinear program is not included here.

2.2.3.3 Variable Tail Link Length

To formulate a trajectory optimization problem where the tail link length is variable, several modifications are made from the original trajectory optimization problem (2.13).

First the link lengths, $L = \{l_i\}_{i=1}^{n_l}$, are included as decision variables. The system dynamics become a function of both the system states and the link lengths, so does the function g for preventing self-collisions. Note that, the mass, moment of inertia of each link is correlated to the link length. Meaning, any change made to the link length result in the change in the mass, and moment of inertia.

To ensure the optimization results are comparable with those in which the length of each individual link in a model is uniform, we introduce a constraint to maintain the sum of the link lengths constant. Furthermore, each link length is assigned a positive lower bound to preclude negative values. Here, we set this positive lower bound to 0.2 m. This choice is made to avoid the necessity of adding extra constraints for self-collision avoidance, which would be required to account for potential self-collision between the tail links with variable lengths, and scenarios where spheres over-approximating the joint connections might not intersect with the torso, yet the portion of the link between the spheres could collide with the torso. While it is possible to employ more comprehensive self-collision constraints, such as convex polytopes [75], for more nuanced collision avoidance, the substantial increase in implementation complexity and the potential increase in computational time render them unnecessary for this study. It is also noteworthy that a 0.2 m lower bound is smaller than the uniform link length of a model with a six-link tail, thus allowing the exploration of shorter links not considered in the original optimization problem.

With these changes, the formulation of the trajectory optimization allowing varied tail link lengths is as follows,

$$
\begin{aligned}
\min_{x,u,\dot{u},L} \quad & \int_{t_0}^{t_f} \| Cx(t) - \Theta(t) \|_2^2 \, dt & \\
\text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t), L) & \forall t \in [t_0, t_f] \\
& S_{lb} \le x(t) \le S_{ub} & \forall t \in [t_0, t_f] \\
& U_{lb} \le u(t) \le U_{ub} & \forall t \in [t_0, t_f] \\
& R_{lb} \le \dot{u}(t) \le R_{ub} & \forall t \in [t_0, t_f] \\
& u(t)^\top u(t) \le E & \forall t \in [t_0, t_f] \\
& g(x(t), L) \le O_{n_g \times 1} & \forall t \in [t_0, t_f] \\
& L \ge H_{lb} & \\
& \sum_{i=1}^{n_l} l_i = \Gamma &
\end{aligned}
\tag{2.16}
$$

We also transform this trajectory optimization problem into a nonlinear program. For brevity, we omit the specifics of the nonlinear program.

2.3 Results and Discussion

This section details the implementation and subsequent analysis of optimization solutions. Given the nonlinear programs outlined in the preceding section, we employed MATLAB's FMINCON as the solver for these nonlinear programs. To enhance computational efficiency, the calculation of model dynamics, which constitutes a significant portion of computation time, was performed using the Roy Featherstone algorithm [47], recognized for its efficiency in calculating rigid-body dynamics. To improve solution optimality, analytical derivatives were provided for all involved constraints. Regarding the discretization of continuous-time trajectory optimization over 0.5 s, a time step dt of 0.004 s was selected, balancing precision with computational feasibility.

Given the nonlinear nature of the programs, well-chosen initialization can significantly improve the solver's effectiveness. Conversely, a poor initial guess may lead to the solver getting trapped in a suboptimal local minimum. To navigate this, we introduced five distinct initial conditions for each program. For programs where the length of each link segment in the tail of model is uniform, the initial conditions include: zero states and zero control inputs, which, although simplistic, meet all constraints, including the dynamics constraints; a straight line in state space between the initial and final states for the torso orientation trajectory, a commonly used initialization strategy; and three other randomly generated initial conditions within the system state and control input boundaries. For programs optimizing over tail link segment lengths, we applied the same five distinct initial conditions for the states and control inputs. For the length decision variables, we used uniform link lengths as the initial guess.

Upon applying these various initial conditions, we assessed the results based on the cost associated with the objective function, selecting the "best" solution characterized by the minimum objective function value.

We validated the solution for each program by forward simulating the model dynamics in MATLAB using ODE45, applying the control inputs derived from the optimization problem, and confirmed that the simulated states match the states derived from the optimization solutions, which provides a solid basis for the findings and conclusions drawn in subsequent subsections.

2.3.1 Uniform Tail Link Length

Given a desired torso orientation trajectory, to quantify the tracking performance of a model, we utilized the value of the objective function as defined in (2.14), namely, the integral of squared error, presented in degrees squared. As illustrated in Fig 2.3, the tracking error decreases as the number of links increases. Specifically, the mean tracking error across all designed torso orientation trajectories used in the optimization problems is 333.30, 214.80, 170.54, 150.94, 137.17, and 129.05 deg^2, for tail configurations ranging from one-link to six-link, respectively. Compared to the single-link tail case, the six-link tail configuration improves tracking accuracy by 61.28%.

This enhanced tracking accuracy demonstrates that an increase in the number of links improves the model's ability to trace a maneuverable trajectory, thereby suggesting that a multi-link tail enables the torso to generate more maneuverable behaviors. Thus, the introduction of articulation in tails can enhance inertial maneuverability.

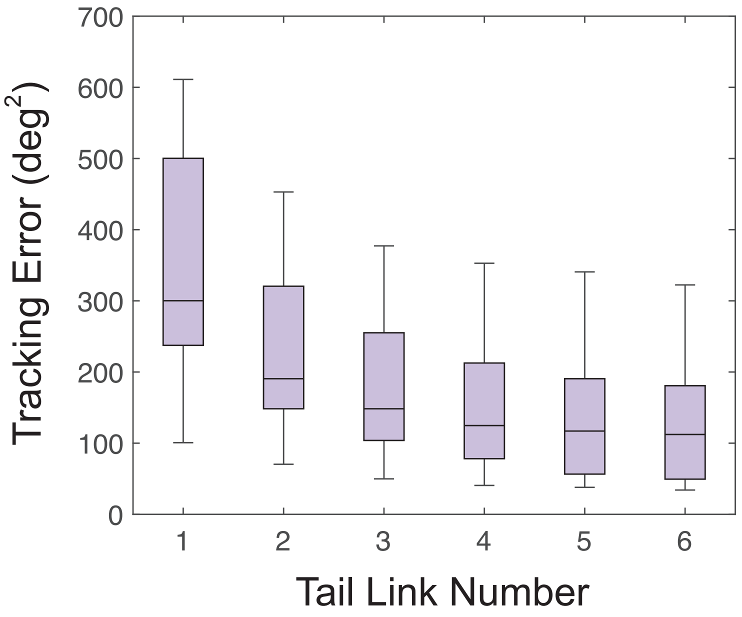

Figure 2.3: **Tracking error comparison.** The box plots display the variability in tracking error for models with tail configurations ranging from one to six links.

While the tracking error consistently reduces with the increase in the number of links, the magnitude of improvement diminishes. This observation aligns with expectations considering the physical constraints imposed on the systems, in particular, the total control effort limit. However, it is important to note that, in the real world, incorporating additional articulation into the model increases the design, modeling, and control complexity. The observed diminishing returns in error reduction underscore a critical point: increasing the number of links is not invariably a better strategy. There might exist a balance point between achieving high articulation and maintaining manageable complexity in design and control, which is pivotal in enhancing the model's maneuverability without overly complicating its construction and operation.

2.3.2 Extended Reaction Wheel

ERW models exhibit better tracking performance than the single link tail model (see Fig 2.4). The average tracking error of the ERW model, corresponding to the two-link tail model, is 13.7% lower than its respective. However, the ERW models do not surpass the performance of tail models with tails comprising three or more links. Although the size of the reaction wheel in the ERW models increases correspondingly with the number of links in the tail of comparable tail models—meaning a larger moment of inertia—this increase does not enhance maneuverability in terms of tracking error. This is likely due to the difficulty of changing directions for a larger wheel, and the link connecting the wheel to the torso, which affects maneuverability. Overall, the tracking performance of ERW models falls between that of the single rigid link tail model and jointed tail models.

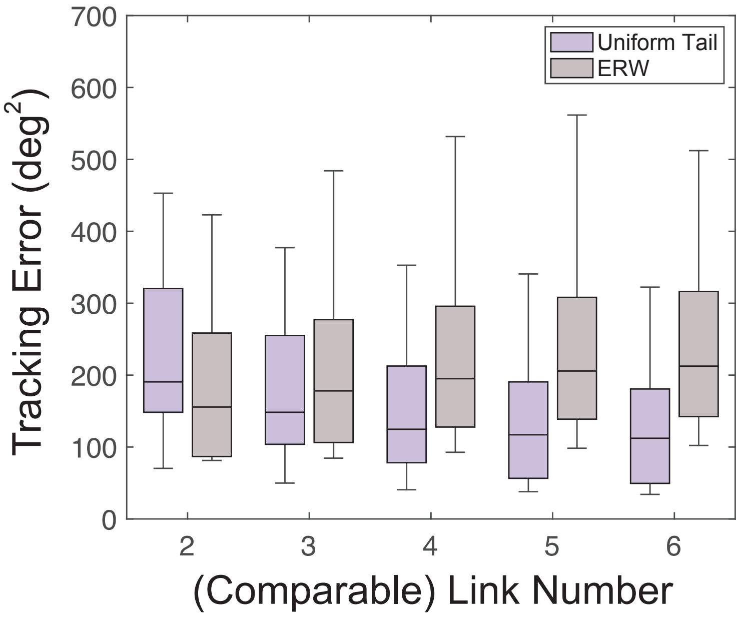

Figure 2.4: **Tracking error comparison.** The box plots display the variability in tracking error for models with tail configurations ranging from one to six links, and ERW models that correspond to the tail models of two to six links.

2.3.3 Variable Tail Link Length

As discussed in 2.2, for tails composed of non-uniform link segment lengths, we limited our analysis to configurations of up to four links due to the escalating computational challenges involved in exporting symbolic dynamics expressions that accommodate variable link lengths. The comparison of tracking errors between the scenario where each tail link segment maintains a constant length and the scenario with variable lengths is depicted in Fig 2.5.

Upon comparing models with uniform tail link segment lengths to those with variable lengths, it is notable from Fig 2.5 that variable link lengths contribute to a further reduction in tracking error. Specifically, the mean tracking errors for configurations with two, three, and four links are 202.15, 158.55, and 140.39 deg^2, respectively. This represents, on average, a 6.7% improvement in tracking accuracy compared to the respective models with uniform tail link segment lengths, highlighting the potential benefits of incorporating non-uniform link lengths in robotic tails designs to enhance maneuverability.

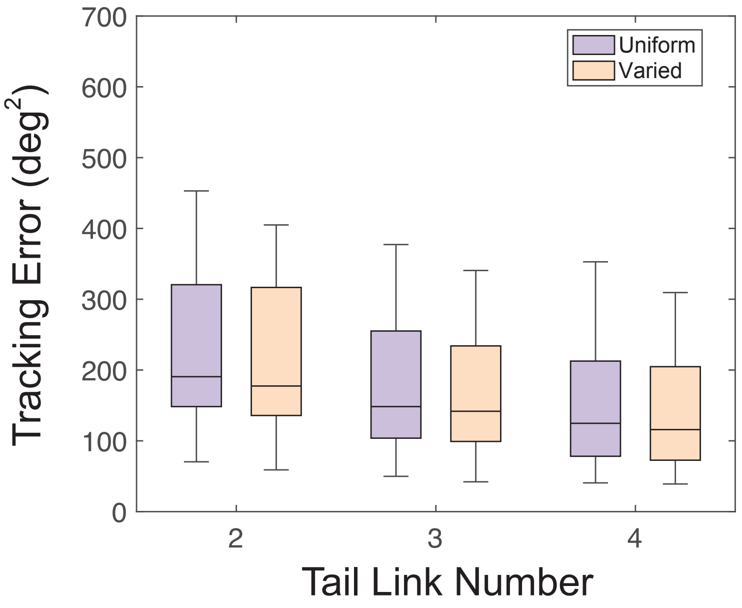

Figure 2.5: **Tracking error comparison.** This box plot demonstrates the variability in tracking errors across robotic tails with configurations of two to four links. It contrasts uniform link segment tails with those whose link sizes have been optimized, highlighting differences in tracking performance.

More interestingly, the length distribution of the tails exhibited a discernible pattern. As observed in Fig 2.6, the first link—the one directly connected to the torso—is typically the shortest among all the links. The second link, conversely, tends to be the longest. For configurations involving three and four links, the length of the links decreases after the second link, creating a pattern where the mid-section area, or more specifically, slightly cranial to the mid-section, of the tail tends to be longer than the other sections. This pattern, unexpectedly, matches the one observed in the tail vertebrae of some animals known for their significant tail involvement in achieving high maneuverability. This similarity likely suggests that such a pattern may contribute to the exceptional maneuverability of these animals and could provide valuable insights for future robotic tail designs.

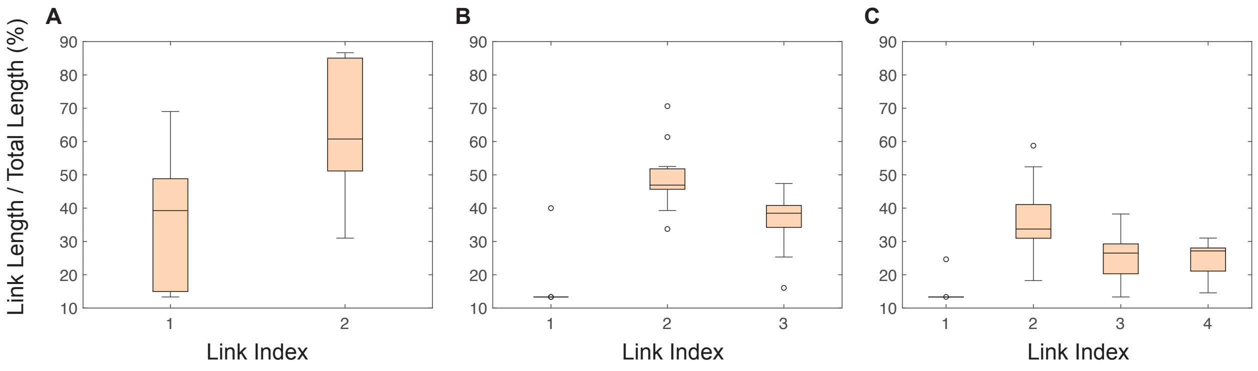

Figure 2.6: **Optimized link lengths distribution.** Variability in optimized link lengths for models with different tail configurations: (**A**) two-link tail, (**B**) three-link tail, and (**C**) four-link tail.

2.4 Conclusions

In this study, we compared the capabilities of different types of inertial appendages and their contributions to three-dimensional body rotation using physics-based simulations and trajectory optimization techniques. These diverse inertial appendages include a single-link pendulum tail, jointed tails, and a type of variant of reaction wheels.

We found that, under certain physical constraints, jointed tails outperformed the other inertial appendages in terms of inducing body rotations by actuating the appendages. Furthermore, the greater the number of links in the jointed tails, the larger the induced body rotation.

Additionally, in the part of the study where we optimized tail segment lengths, we found that varying the segment lengths further enhances three-dimensional body rotation. Subject to the constraint of a fixed total tail length, we observed a pattern in the length distribution of tail segments:

the lengths tend to be smaller near the base and the tip of the tail, with larger segments around the midsection. Interestingly, this pattern closely resembles that observed in the tails of animals known for their rapid large body rotation capability.

Due to the significant intricacies involved in the dynamics, this study has not comprehensively explained why the vertebrae in these animals exhibit such a pattern. One preliminary hypothesis regarding the reasons behind such a pattern—particularly from a moment of inertia perspective—is that lower mass near the base (proximal end) and at the tip (distal end) allow these sections to accelerate and decelerate more rapidly. This can facilitate quick initiation and cessation of movement. The longer links in the mid-region increase the tail's mass in this section, making it better suited for generating forceful movements. This is because a larger moment of inertia allows the mid-section to store more rotational energy, contributing to powerful swings or thrusts that can propel the animal through its environment or enable swift directional changes.

The resemblance in length distribution suggests that this pattern may contribute to their impressive capability for rapid and large body rotation and may offer valuable insights for future robotic tail design. In particular, for those interested in developing robotic tails to enhance the inertia maneuvering capabilities of robots, it would be beneficial to consider optimizing the mass distribution within the robotic tail. This is particularly relevant when working within a certain level of design complexity and control for an articulated tail, as it can further enhance the inertia maneuvering performance.

Overall, our findings suggest that a certain level of tail articulation is beneficial for large body rotations. They motivate us to further investigate this area to uncover the underlying mechanisms and inform the design of robots with tails.

CHAPTER 3

`ArborSim`: Articulated, Branching, OpenSim Routing for Constructing Models of Multi-jointed Appendages with Complex Muscle-tendon Architecture

3.1 Introduction

Animal movement results from intricate interactions among elements of musculoskeletal systems [45, 53, 88]. Investigating these interactions using experimental in vivo data alone can be challenging and often impractical due to measurement constraints and invasive methods. Computational musculoskeletal modeling and simulations offer a complementary approach. By integrating models of system elements, dynamic simulations utilizing the integrated model are generated, providing estimates of hard-to-measure variables [39, 133]. These simulations enable researchers to reproduce observed experimental results, comprehend system coordination, design innovative experiments, and predict treatment outcomes, making them valuable tools in biomechanics studies [39, 133].

In the last few decades, substantial advancements have been made in detailed musculoskeletal modeling of the human body [44, 51, 60, 10, 110, 117, 121], which has led to the discovery of fundamental human movement principles, advanced injury diagnosis, and informed prosthetic and robotic design [95, 49, 76, 70, 59, 128, 94, 124, 11, 46, 137]. OpenSim [41, 123], a widely-used open-source software for musculoskeletal modeling and simulation, has been a key contributor to these advancements, fostering the sharing and dissemination of musculoskeletal models among the biomechanics community through the SimTK database. Recently, there has been increasing interest in developing musculoskeletal models of other animals [98, 62, 35, 38, 127, 111], such as ostrich [62], chimpanzee [98], and mouse [35, 111], with the aim of enriching the understanding of a wide range of forms and behaviors exhibited by other species, and inspiring the development of bioinspired robotic technologies [143, 118, 63].

In every vertebrate animal, highly jointed systems, such as phalanges and the axial skeleton (spine, including necks and tails), facilitate greater curvature and are crucial to performance and survival. For example, cats adjust the stiffness of their flexible backs to absorb kinetic energy upon landing, allowing their limbs to attenuate landing impulses more safely and effectively [147]. Certain monkeys use their prehensile tails as a fifth limb, providing essential grasp and balance while navigating through trees, facilitating climbing in their arboreal environment [57]. As an extreme example, snakes are remarkable, highly jointed systems by themselves. They possess highly flexible and muscular bodies that allow for a range of movement adaptations, enabling them to effectively navigate various terrains and tight spaces [93, 1, 12].

Nevertheless, in most existing musculoskeletal models, highly jointed systems are often simplified or excluded to facilitate computationally challenging analyses [145, 18, 127]. For instance, a musculoskeletal model of the dog does not consider the motion of the spine as a series of individual vertebrae, but rather simplifies the spine into discrete regions such as thorax and tail which are granted three degrees of freedom (DOFs) each [127]. A mouse model simulating a trotting gait focuses only on the hindlimb, disregarding the spine and tail entirely [35].

The practice of simplifying highly jointed systems, while helpful in reducing computational complexity, may unintentionally mask their complete biomechanical importance. However, even within these limitations, recent studies using simplified, but articulated models of highly jointed systems have identified a significant role for them in animal locomotion. For example, using a musculoskeletal model of a dinosaur that includes a simplified two-link tail, researchers have found that the tail performs a critical role in enhancing locomotor efficiency [17]. Moreover, by incorporating tail compliance properties through a five-link tail model, researchers have used the natural frequency of a tail to estimate the preferred walking speed of a dinosaur [135]. These studies exemplify the unexpected and substantial contributions of highly jointed systems that await discovery through modeling, underscoring the potential for gaining even more profound insights from more detailed and comprehensive models of these systems.

Despite the development of musculoskeletal modeling software over the past few decades, constructing detailed models of highly jointed systems remains both time-consuming and challenging. The primary challenge results from the presence of numerous bones and intricate muscle and tendon architectures, in which muscles and tendons can span multiple joints; the very morphology that allows highly jointed structures to form smooth curves. To compound upon this complexity, tail tendons frequently show a branching morphology that manifests as multiple tendons arising from a shared muscle-tendon transition zone, tendons splitting distally into multiple branches before reaching their insertion sites on multiple vertebrae, and/or tendons forking near their insertion site to blend into the muscle-tendon transition zone of a small local intrinsic muscle slip [115, 91, 92]. This branching phenomenon greatly challenges musculoskeletal modeling processes that were developed for studying limb mechanics.

When designing software to digitally represent morphology, simplifying assumptions can inadvertently become restrictive constraints. This is evident in the modeling of complex muscle-tendon branching architectures. Due to the limitation of representing only one tendon per muscle, researchers typically adopt an approach utilizing multiple muscle-tendon units (MTUs) in parallel, each representing a unique insertion and a portion of the muscle associated with that insertion. For example, the extensor digitorum communis (EDC) muscle in humans has a tendon that divides into four distinct branches, each of which inserts into a different finger [74]. In existing musculoskeletal models of the human hand [60, 80, 87], the EDC is represented as multiple independent compartments, each associated with a specific tendon that extends to a particular finger. This modeling simplifies the representation of the muscle and tendon architecture. However, it may not fully capture the physical interactions that occur between the muscle and tendons, increasing the likelihood that such models may provide limited, or even misleading predictions.

Physical interactions between branched MTUs are likely to be particularly important in cases where the branches insert on different bones, the MTU spans many joints, or the MTUs form a complex branching network with more than two insertions. For example, consider an MTU that originates on one bone and then inserts on two different bones. If external forces bend the more distal joint, that branch of the tendon experiences stress and strain, which increases stress on the tendon and muscle proximal to the branching site, but reduces stress on parallel distal branch. This type of complex stress field has been essential to predicting tears in the human supraspinatus and infraspinatus [8, 90], but there are currently no examples of OpenSim models that reflect these empirical data. Here, we introduce a new modeling framework to explicitly model the mechanics of branched MTUs and use simplified case studies to estimate the effect of modeling strategy on motion predictions.

Current tools designed for musculoskeletal modeling are not well suited for handling the aforementioned modeling complexities for highly jointed systems. NMSBuilder, an open-source software to create subject-specific musculoskeletal models, was established to facilitate the model building process for lower limbs [134]. The resulting models can be easily exported to OpenSim. While NMSBuilder improves musculoskeletal modeling efficiency, the modeling flow still requires manual addition of every entity and is, therefore, not well-suited for modeling highly articulated systems, particularly those featuring complex muscle-tendon branching architectures. A similar challenge exists with OpenSim Creator [68], a newly introduced software still in its alpha stage, designed for creating musculoskeletal models. Furthermore, NMSBuilder only allows the use of a subset of modeling components of OpenSim, any elements absent in NMSBuilder, such as constraints and controllers, must be added manually post hoc in OpenSim if required. Other modeling software options, such as SIMM [42, 43], Anybody [112], and Visual 3D (Visual 3D Professional, C-motion, Germantown, MD, USA), are proprietary, expensive ($4995 – 7950 per year, with academic discounts applied), and tailored for limb modeling. Because none of these software packages provide full access to source code, it is difficult to expand their functionality to accommodate the complex tendon branching phenomena. Here, we introduce `ArborSim` (Articulated, Branching, OpenSim Routing), an extensible open-source tool integrated with the application programming interface (API) of OpenSim for constructing detailed musculoskeletal models of highly jointed systems, while accounting for complex muscle-tendon architectures. We intend the toolkit to enable biomechanics researchers to efficiently construct musculoskeletal models of highly jointed structures from CT scans and dissection data. We provide templates to store all essential data for modeling as Comma-Separated Values (CSV) files that are then used to automatically construct the model. This highly customizable framework enables users to easily set or update variables based on empirical data for any musculoskeletal parameter, even in highly complex systems.

In addition, we propose a novel approach to modeling complex muscle-tendon architectures that explicitly considers the interactions between each component. The approach is seamlessly integrated into the tool. By changing one line of code, users can decide to construct musculoskeletal models using either the conventional (parallel MTU) approach to modeling branching or our proposed method, both using the same data. We employ our tool to investigate the impact of using different methods to model MTU branching by constructing simplified, multi-jointed systems, i.e. "toy models." Large differences between the kinematic outputs of simulations using the conventional branching modeling method and the proposed method highlight the need for more accurate models of complex branching phenomena.

3.2 Methods

3.2.1 Overview of `ArborSim`

A fundamental representation of a musculoskeletal system in OpenSim involves rigid bodies, joints, and MTUs. The rigid bodies represent bones and embody associated inertial characteristics including mass, center of mass, and moment of inertia that incorporate the mass of the muscles and tendons that move in tandem with each bone. These rigid bodies are interconnected through joints defined by Joint Coordinate Systems (JCSs). MTUs, functioning as actuators, are represented by lines of action along with a set of parameters that govern muscle contraction dynamics. These lines of action are determined by MTU paths, which consist of user-selected path points. Additionally, wrap objects can be introduced to shape the line of action of an MTU along parametric surfaces, commonly known as wrapping surfaces.

The development of our tool centers around the procedural creation of these fundamental entities. To enhance the software's structure, reusability, and ease of maintenance, the tool is built using a four-layer architecture (see Fig 4.5). Among the four layers, the first and second layers (Fig 4.5A-B) serve as input interfaces, allowing users to input customized data for constructing an articulated musculoskeletal model of a highly jointed system. In these two layers, the first requires data in a "global" format, whereas the second needs "local" format data (More details on these formats are provided in subsequent sections). Users have the flexibility to input data in either format. If "global" data is provided, the "**Transformer**" feature will convert it into "local" format, ready for model construction.

The components of the musculoskeletal model, derived from the inputs of the first two layers, are then modified via the implementation of the proposed branching modeling approach, named "**Brancher**". These modified components reside in the third layer (Fig 4.5C). During this modification process, branched muscles and tendons–initially represented as multiple independent MTUs according to the conventional branching modeling method–are altered to conform with the proposed branching modeling method. Subsequently, depending on the desired branching representation, the tool assembles the stored components either in the second or third layer via the functionality named "**Builder**", thereby generating and exporting the resulting musculoskeletal model that can be directly loaded into OpenSim. This resulting model constitutes the fourth layer (Fig 4.5D).

Each category of data within the first two layers is stored in a dedicated CSV file. The natural data structure and ubiquity of CSV files offer a clear and intuitive way to represent the data, and simplify data sharing and collaboration with researchers using diverse tools or platforms. More-

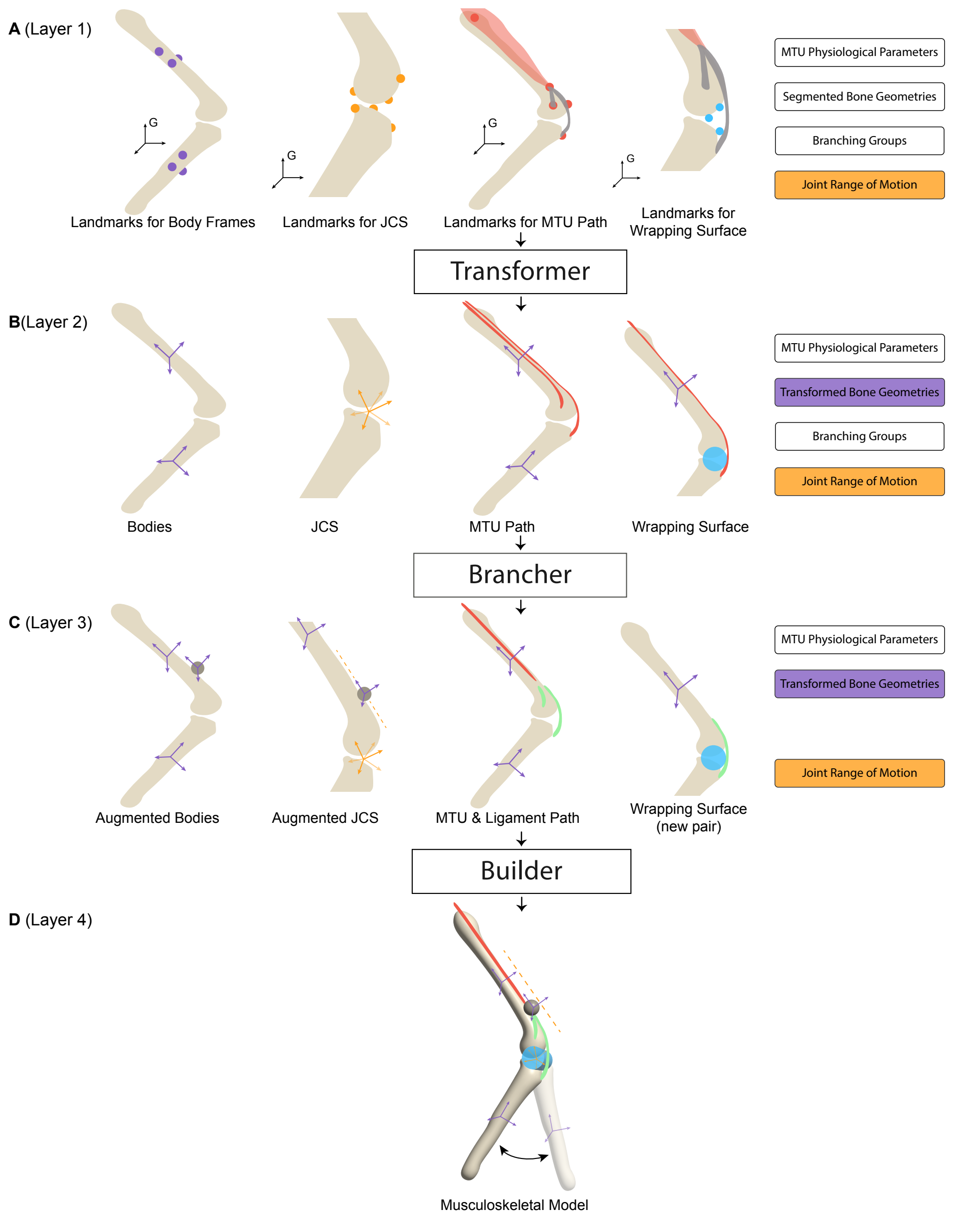

Figure 3.1: **Overview of `ArborSim`.** (**A**) The input layer with global data, divided into eight categories. In the diagrams, purple dots indicate anatomical landmarks used for establishing body frames. Orange dots represent landmarks for defining JCSs, while red dots denote landmarks for MTU paths. Blue dots are used to mark landmarks for defining wrapping surfaces. Transparent red contours depict the muscle, grey slender contours outline the tendons, and the black frames labeled "G" represent the reference frame in which the landmark coordinates are collected. (**B**) The input layer with local data. Separate MTUs (red) are used to represent the muscle-tendon structure. (**C**) The middle layer with modified components following the proposed branching modeling method. The dark grey circles denote the extra body introduced to connect the single MTU (red) and the passive elements (green). (**D**) The output layer with the constructed musculoskeletal model.

over, the tool can be easily extended, as one can create additional dedicated CSV files to accommodate extra components and functionalities for the tool.

3.2.2 Layer 1

In the first layer, users provide the following categories of data:

- Coordinates of anatomical landmarks for establishing body frames.
- Coordinates of anatomical landmarks for defining JCSs.
- Coordinates of anatomical landmarks outlining paths for MTUs.
- Coordinates of anatomical landmarks for wrapping surfaces.
- Physiological parameters of MTUs.
- Segmented bone geometries.
- Branching groups.
- Joint range of motion.

Among the data mentioned above, the coordinates of anatomical landmarks are represented in a global reference frame. In the context of working with a CT scan or other biomedical imaging resources, these landmarks can be identified in segmentation and visualization software such as Mimics (Materialise NV, Leuven, Belgium), 3D Slicer [48], and ITK-SNAP [146]. In this context, the term "global" coordinates of these landmarks refers to the coordinates within the default coordinate system in which the CT scan is saved and visualized. This default coordinate system provides a common and consistent spatial reference for all the data in the CT scan.

To establish a reference coordinate system within the current software setup, it is required to provide three landmark points: one for the origin of the coordinate system, one to denote the positive X-axis direction, and another to denote the positive Y-axis direction. Thus, for each body, users provide coordinates of three anatomical landmarks for establishing the corresponding body frame.

To create a joint that defines the kinematic relationship between two bodies, termed parent and child bodies, two joint frames are constructed, each affixed to the parent body and child body (see Fig 3.3), respectively. Therefore, six landmark points are required to construct the joint frames for each joint. Three of these landmarks specify the location and orientation of the joint frame on the parent body, and the other three specify the location and orientation of the joint frame on the child body. During the JCS landmark identification process, some recommended guidelines for standardized JCS definitions can be found in [30, 140, 141].

To maintain compatibility with the conventional method of modeling branched muscles and tendons in cases of complex branching phenomena, and to seamlessly incorporate the proposed branching modeling method, offering flexibility in creating models using either technique, the MTU-related data provided in the input are stored in a manner that presents branched muscles and tendons as distinct, separate, and independent muscle-tendon compartments. This adheres to the rule of a single tendon per muscle in OpenSim. For instance, when a muscle is attached to a tendon that branches into two insertion points (see Fig 4.5), two MTUs are used to represent this muscle-tendon structure. The anatomical landmarks for MTU path are provided for each MTU, featuring identical landmarks along the muscle pathway, and different landmarks along the tendon pathway. Correspondingly, two sets of physiological parameters are designated for these two MTUs, encompassing divided maximal isometric force, consistent pennation angle and optimal fiber length, and different tendon slack length, etc..

To enable MTUs to interact with certain wrapping objects, mimicking the way muscles and tendons navigate around bones and other structures, wrapping surfaces can be established. These surfaces guide the paths of linked MTUs in a manner that cannot be achieved with fixed MTU path points alone. To create a wrapping surface, provide three anatomical landmarks that define the reference frame, as well as additional geometry parameters such as the radius and height of the wrapping surface.

To prepare for implementing the proposed branching method, i.e., the conversion from the second layer to the third layer, the fact that multiple MTUs together form a branching unit within the biological equivalent is preserved as a branching group data.

For other essential data, the segmented bone geometries enable effective visualization within OpenSim. Notably, these segmented geometries are exported directly from the same global reference frame used for collecting landmark coordinate data, aligning with the default frame of the CT scan or other imaging data.

The joint range of motion data restricts the relative movement of the bodies. Such data might not be immediately accessible to users until the joints are precisely defined in the model as this information is more readily obtainable in local frames as opposed to a single global frame. Therefore, users may initially overlook this aspect and later assign joint range of motions after constructing a model devoid of embedded joint range of motion. In a model, joint range of motion can be determined by ensuring no penetration occurs between the parent and child body of a joint. Experimental data can also be instrumental in this process.

3.2.3 Transformer Creates Layer 2

In contrast to data in the first layer, which is collected under the same global coordinate system, OpenSim models use distinct reference frames for each body, making the model more biomechanically relevant and intuitive. Consequently, it necessitates the transformation of each segmented bone geometry, initially exported from the segmented CT scan, so that each body in the resulting model has its own local coordinate system. Additionally, elements such as joints, MTU paths, wrapping surfaces, etc., are constructed based on the relationships between associated bodies. This construction requires transforming their "global" coordinates into the respective "local" coordinate systems of the associated bodies.

Within the tool, when the "global" data and other necessary data, are input into the first layer, the tool seamlessly performs the required transformations from the "global" data to a "local" format, while keeping other necessary data unchanged. From there, users can obtain bodies, JCSs, MTUs, and wrapping surfaces that are directly compatible with OpenSim, all of which are stored in the second layer. This conversion is particularly advantageous as it eliminates the need for manual transformations by users. Consequently, users can directly integrate data obtained from other software tools, thus streamlining the model development process.

A detailed explanation of the automated transformation from "global" to "local" is provided as follows.

As stated in previous subsection, to establish a reference coordinate system, three landmark points are provided. These three landmark points combine to form two vectors, representing the reference X and Y-axes (see Fig 3.2), respectively.

When identifying these landmarks, it is very likely that these two vectors are not perfectly perpendicular to each other due to various factors, such as measurement errors during the process. This does not conform to standard Cartesian geometry. To address this, the tool adjusts the location of the landmark indicating the positive direction of the Y-axis. This is achieved by locating a new point on the plane defined by these two vectors that is closest to the original landmark for the positive Y-axis direction, and ensuring that the new point creates a resulting angle of 90 degrees between the reference X-axis and corrected reference Y-axis. Subsequently, the reference Z-axis can be defined by taking the cross product of the X-axis vector and corrected Y-axis vector. Finally, we normalize these vectors by dividing each by their respective magnitudes to establish a well-defined coordinate system.

By following the procedure outlined above, body frames are established using the "global" coordinates of three anatomical landmarks for each body from the supplied data in the first layer of the tool. From the normalized vectors, which represent body frame axes, we can build transformation matrices. These transformation matrices can then transform the original segmented bone geometries from the "global" coordinate system to their individual "local" coordinate systems. As a result, with the transformed bone geometries and associated center of mass and inertia defined in the corresponding body frames, a set of rigid bodies–a primary building block of the model–can be readily derived using OpenSim's API.

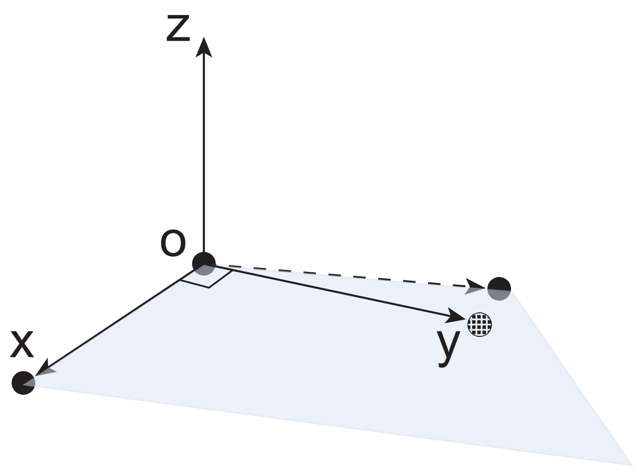

Figure 3.2: **Coordinate system construction.** Three solid black dots indicate the landmarks for the origin and the positive directions of the X and Y axes, respectively. The plane, depicted in light blue, is defined by these points. The dashed black dot represents the adjusted landmark for the positive Y-axis direction, ensuring orthogonality with the X-axis.

With six anatomical landmarks provided for each joint construction, as explained earlier, three represent the joint frame's location and orientation on the parent body, and the other three represent those on the child body. Following the method used to transform bone geometries from "global" to "local", a constant transformation that represents the offset between a joint frame and the corresponding body frame can be calculated. That establishes the JCSs.

Joints are modeled as CustomJoints, which is the most generic joint representation in OpenSim. It allows for motion in 6 DOFs, offering the flexibility in modeling traditional joints (e.g., slider joint, universal joint) as well as complex biomechanical joints.

Implementing the same transformation concept, the positions of MTU path points relative to the body frames through which the MTU traverses can be calculated. Similarly, the location and orientation of a wrapping object in relation to the associated body segments can also be determined.

The aforementioned section explains how the tool efficiently converts the provided "global" data into a "local" format, designed to integrate smoothly with OpenSim. Alternatively, if users opt to directly provide local data and construct a model from there, they can bypass the first layer and simply supply the local data through the second layer. This feature allows users to create artificial

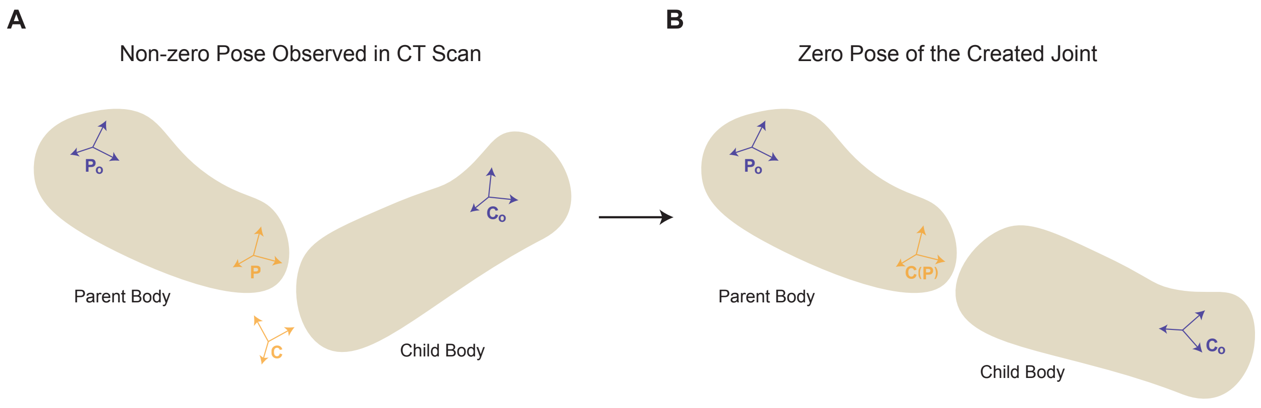
A
Non-zero Pose Observed in CT Scan
Po
P
C
Co
Parent Body
Child Body
B
Zero Pose of the Created Joint
Po
C(P)
Co
Parent Body
Child Body

Figure 3.3: **Joint construction.** A joint that defines the kinematic relationship between frames P and C is created, where frame P is affixed to the parent body, with its body frame denoted by P_o, and frame C is affixed to the child body, with its body is frame denoted by C_o. By making frames P and C coincident, the non-zero pose observed when collecting landmarks to build the JCS from CT scans or other data sources, shown in (**A**), is transformed into the default zero-pose, shown in (**B**).

toy musculoskeletal models conveniently, as working with local data aligns more intuitively with the understanding of the desired local relationships.

3.2.4 Brancher Creates Layer 3

As briefly touched on in previous sections, the traditional representation of a muscle-tendon branching structure involves multiple independent MTUs (see Fig 4.5B). In that representative scenario, two distinct MTUs are used to account for the two tendons with different insertions. Correspondingly, the muscle is represented as two compartments, each associated with a tendon. Both MTUs stem from the same muscle origin, following the same path up to the branching point, then diverging along their unique tendon paths.

The physiological parameters of the two MTUs depend on the respective muscle compartment and tendon. Usually, they share the same muscle pennation angle, and possess nearly identical optimal fiber lengths. The maximal isometric force of the muscle is allocated based on the physiological cross-sectional area (PCSA) of each muscle compartment, and the tendon slack length of each MTU aligns with the tendon's resting length. Collectively, these parameters characterize the active and passive force-length curve, the force-velocity curve of the muscle, and the force-length curve of the tendons, determining the force development of MTUs in a musculoskeletal model. To synchronize the functioning of these muscle compartments, two identical muscle excitation signals can be applied across the MTUs.

In contrast to the conventional method, our proposed branching modeling approach utilizes a single MTU to represent the segment of muscles and tendons that share the same pathway from their origin, prior to the emergence of any branching, within a group of original MTUs specified in the provided branching group data. Hence, in the representative scenario (Fig 4.5C), the muscle is no longer fragmented into two compartments. Instead, a single MTU is used to represent the muscle in the real physiological system. The maximal isometric force of this single MTU equals the sum of the maximal isometric forces from the two MTUs in the conventional branching modeling method, while its optimal fiber length and pennation angle remain consistent with those of the two MTUs.

There are two cases of tendon branching that must be modeled slightly differently. The muscle-tendon unit depicted in Fig 4.5C directly branches into two tendons at the muscle-tendon transition. In this case, the single MTU would ideally represent a muscle without tendons. However, because OpenSim does not support MTUs with zero tendon (slack) length, an exceedingly short tendon can be introduced into that single MTU. This ensures that the MTU predominantly represents the muscle. To prevent the stretching of this extra tendon from affecting muscle force distribution and coordination to the downstream tendons, the short tendon can be made rigid. Alternatively, when a tendon segment is connected to the muscle before branching occurs, the additional rigid short tendon becomes unnecessary. Instead, we can simply use an MTU to represent the muscle associated with that particular tendon.

To represent the tendons post-branching, ligaments are used. In OpenSim, as in biomechanics in general, a ligament represents passive connective tissue that connects bones to other bones. Both the tendon in an MTU and a ligament are defined by a normalized force-length curve that describes how the force responds to being stretched, a force scale that scales the normalized force-length curve, and the resting length. Therefore, by assigning the same properties, they are treated equivalently in OpenSim.

To emulate the branching architecture, extra bodies, modeled as spheres, are introduced for each branching point. As branching points in reality can slide relative to vertebrae, a CustomJoint is incorporated between each extra body and its corresponding vertebra to ensure this movement capability. Within the CustomJoint, the three rotational DOFs are restricted, rendering it solely as a sliding joint. To effectively negate the impact of inertia changes induced by the inclusion of the extra body to the model on the force development and the overall motion, the mass of the extra body is set to an exceptionally low value. For the representative muscle-tendon structure, one extra body is employed at the muscle-tendon transition point, accounting for the muscle dividing into two tendons. By linking the extra body to the respective tendon insertion points using ligaments, the muscle and tendons in the musculoskeletal system are inherently interconnected in the model. Therefore, the intricate interactions and dependencies within the musculoskeletal system can be more faithfully represented.

In the proposed branching modeling method, the resting length of each ligament should match that of its corresponding tendon segment in the actual system. The force scale for each ligament should be allocated as a portion of the muscle's maximal isometric force, because the force in the normalized force-length curve is normalized by the muscle's maximal isometric force in OpenSim. If the tendons have uniform material properties, the ligament's force scale can be determined by distributing the maximal isometric force based on the tendons' cross-sectional areas.

In summary, based on the data from the second layer, where the elements of the musculoskeletal model align with the traditional branching modeling approach, the proposed branching modeling method demands several adaptations. Specifically, these include the addition of supplemental bodies to the original model, the designation of sliding joints for these newly added bodies, adjustments to MTU pathway, the integration of ligaments to denote the branched tendons, recalibration of muscle-tendon physiological parameters, and updates to the paired association between the wrapping surfaces and the MTUs/ligaments. This conversion is inherently integrated into the tool, facilitating the creation of models using the proposed branching modeling method. The detailed procedure for the implementation of this conversion is expounded in Algorithm 1.

In the algorithm, directed tree graphs are constructed to represent the muscle-tendon branching structure. By analyzing these tree graphs, the algorithm automatically makes the necessary adjustments and outputs the modified components into CSV files, except for the reassessment of muscle-tendon physiological parameters. Users manually input physiological parameters for the modified MTUs and newly introduced ligaments in the corresponding CSV file, based on anatomical data.

3.2.5 Builder Creates Layer 4

Leveraging OpenSim's API, the modified components in the third layer that fits into the OpenSim are integrated via the functionality named "**Builder**" to create a musculoskeletal model. In this model, the branched muscles and tendons are modeled using the proposed method. As mentioned earlier, the components in the second layer present the branched muscles and tendons using the conventional method, and these components are also directly compatible with OpenSim. Therefore, to create a musculoskeletal model using the conventional branching modeling method, users can simply bypass the third layer and construct a model from the second layer. Thus, the tool offers the capability of creating two musculoskeletal models, using two different branching modeling methods, both utilizing the same data.

3.2.6 Comparative Simulation Case Study

Here, we employ `ArborSim` to investigate the tangible distinctions between the conventional parallel modeling method and the proposed branching modeling method for modeling complex muscle-tendon architectures. The investigation used comparative simulations conducted on artificial toy articulated musculoskeletal systems. Within these systems, we created muscle-tendon architectures with branching features. We focused on the branching of tendons rather than muscles, in line with our findings in mammalian tails [91, 92].

Algorithm 1: Brancher

```
Step 1. Construct Directed Tree Graph
foreach branching group do
    foreach MTU in the branching group do
        foreach path point of the MTU (from its origin to its insertion) do
            Check for existing vertex by associated body, the position on that body, and the
              point type
            if vertex exists then
                Append the path point information (MTU name, index, type) to the vertex
            else
                Create a new vertex for the path point
                Associate the path point information to the vertex
                if path point is not the origin then
                    Create a directed edge from the previous path point vertex to the new
                      vertex
Step 2. Identify Special Vertices
foreach vertex in the graph do
    if vertex's indegree $\times$ vertex's outdegree $\neq 0$ and vertex's indegree $+$ vertex's
      outdegree $>= 3$ then
        Mark the vertex as a branching vertex
    if vertex represents an MTU's origin then
        Mark the vertex as an origin vertex
    if vertex represents an MTU's insertion then
        Mark the vertex as an insertion vertex
Step 3. Add Extra Bodies and Sliding Joints
foreach branching vertex do
    Construct an extra body modeled as a sphere
    Build a CustomJoint with translational DOFs between the associated body of the
      vertex and the sphere
Step 4. Refactor MTU Paths and Build Ligament Paths
foreach origin vertex do
    Traverse the graph until reaching any branching vertex
    Model this segment as an MTU, originating from the origin vertex's associated body,
      inserting to the branching vertex's associated extra body
foreach branching vertex do
    Traverse the graph until reaching a branching or insertion vertex
    Model this segment as a ligament, originating from the branching vertex's associated
      body, inserting to the branching or insertion vertex's associated (extra) body
```

Drawing from observed variations in muscle-tendon branching architectures across a variety of mammalian tails [91, 92], we identified three relatively prevalent categories of variation, which serve as the main focus of our comparative simulations:

- Variations in the number of tendon branches into which a muscle divides.
- Variations in the number of joints between the terminals of the branched tendons.
- Variations in the ratio of muscle fiber length to MTU length.

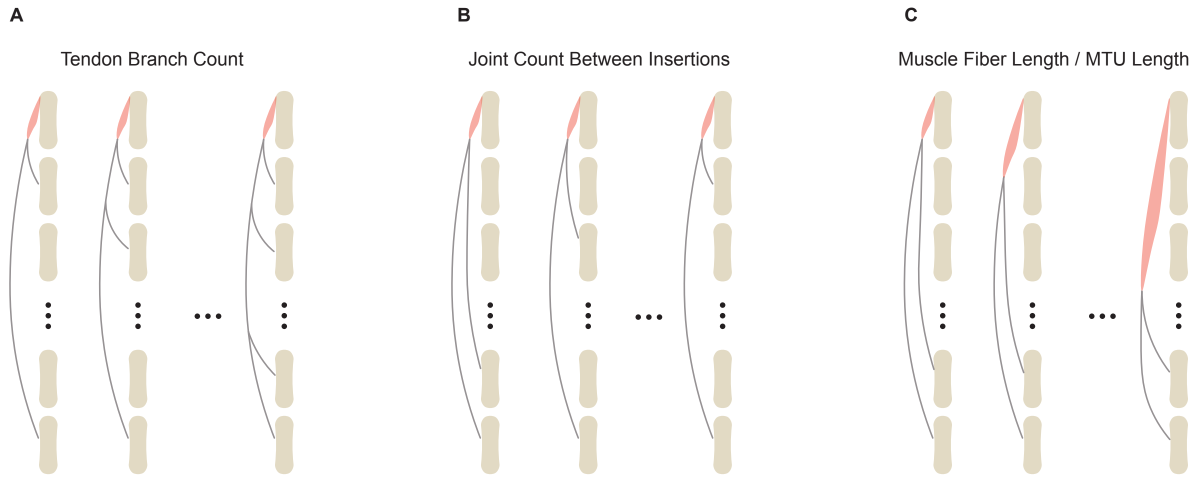
A
Tendon Branch Count
B
Joint Count Between Insertions
C
Muscle Fiber Length / MTU Length

Figure 3.4: **Comparative simulation categories.** The category of varying numbers of tendon branches is shown in (**A**). The category representing the varying numbers of joints spanned between insertions is depicted in (**B**). The category showing the varied muscle fiber length to the longest MTU ratio is presented in (**C**). In all three diagrams illustrating these categories, the red contours indicate the muscles, while the grey slender contours represent the branched tendons. To more clearly visualize the branching, the tendons and muscles in the diagrams are displayed away from the bones. However, in the models, they are all connected to the bones through the use of 'via points' on the bone surfaces, allowing them to exert force on the bodies along the pathway.

These categories are illustrated in Fig 3.4. For each category, six musculoskeletal systems were designed. To simplify the comparison between the traditional branching modeling approach and our proposed method, we restricted these toy musculoskeletal models to a planar configuration. Each toy musculoskeletal model consisted of 8 vertebrae connected by planar joints, with each vertebra modeled as a uniformly dense cylinder in OpenSim. The cylinders had a mass of 0.1 kg, length of 0.2 m, and radius of 0.025 m. A gap of 0.025 m between each pair of consecutive vertebrae was introduced to prevent interpenetration during movement. The range of motion for each joint was calculated based on the cylinder dimensions and the gap length, with ForceCoordinateLimitForce in OpenSim applied to enforce these constraints.

Each system was equipped with a muscle-tendon branching architecture, tailored to its respective category. Each muscle-tendon branching architecture consisted of a single muscle and multiple branched tendons. To ensure that all vertebrae actively respond to muscle activation, a long tendon spanning from the muscle-tendon transition area to the last vertebra (most bottom in Fig) was present in all the systems. In the category where the ratio of muscle fiber length to MTU length varied (Fig 3.4C), the MTU length was denoted by the measurement of the longest MTU in a given model.

For each system, two musculoskeletal models were constructed: one using the conventional branching modeling method and the other using the proposed branching modeling method, following the steps presented in previous subsections (N.B. both branch modeling methods are incorporated within `ArborSim`). Given the physiological parameters of the muscle, for models using the conventional branching modeling method, the maximal isometric force of the muscle was evenly allocated among these MTUs. For models using the proposed branching modeling method–where a single MTU is used to represent the muscle–the maximal isometric force was directly assigned to that MTU. For fair comparison between equivalent models, we divided the total maximum isometric force of the muscle by the number of insertions to set the force scale for each distal ligament element. The force scale for the proximal ligaments was then set to the sum of the force scales of the connected distal ligaments.

Regarding other MTU and ligament modeling parameters, the optimal and default muscle fiber lengths were defined as the distance between the muscle's origin and insertion in a zero-pose, where all the joint angles are zero, resulting in a straight, fully extended pose. Similarly, the tendon slack lengths for MTUs in the conventional method, as well as the resting lengths for ligaments in the proposed method, were established based on the default lengths of their respective segments in this zero-pose configuration. For the branched model, an extremely short rigid tendon of 0.0001 m was included in the MTU that represents the muscle to circuvment OpenSim's limitation of not supporting MTUs with zero tendon length.

To ensure comparability between the conventional and proposed methods, the same normalized force-length curves and force-velocity curves were used for the muscle. Additionally, the normalized force-length curve of the tendon in the MTU was identical to that of the ligament. The muscle was modeled as a Millard2012EquilibriumMuscle [89] in OpenSim. The mass of the extra body introduced when using the proposed branching modeling method was set to 5×10^{-4} kg.

Within each category, an identical muscle excitation signal was applied to all systems with varying muscle-tendon architectures, and simulations were run under that excitation signal for a duration of $T = 3$ seconds. To evaluate the difference between the traditional branching modeling method and proposed method on the resulting kinematics, we defined the following metric:

$$\Delta = \max_t \sum_{i=1}^{n} \frac{|\theta_i^{con}(t) - \theta_i^{pro}(t)|}{\bar{\theta}_i} \times 100\%, \qquad t \in [0, T] \tag{3.1}$$

where $\theta_i^{con}(t)$ denotes the joint angle of i-th joint at time t for models generated using the conventional branching modeling method, $\theta_i^{pro}(t)$ denotes the joint angle of i-th joint at time t for models generated using the proposed branching modeling method, $\bar{\theta}_i$ denotes the positive bound of i-th joint, and n is the number of joints, which is 8 in our study. In the metric, the numerator $\sum_{i=1}^{n} |\theta_i^{con}(t) - \theta_i^{pro}(t)|$ captures the total absolute differences in joints angles across all eight joints at time t. The denominator $\sum_{i=1}^{n} \bar{\theta}_i$ represents the sum of the positive bound of each joint. This reflects the pose where the toy musculoskeletal system is rotated to its limit.

Therefore, the metric quantifies the maximal percentage of the total absolute differences in joint angles across all eight joints over the entire motion trajectory, relative to the joint limits. This metric highlights the worst-case scenario in terms of the discrepancy between the two methods, providing a robust evaluation of their impact on system behavior.

3.2.7 Sensitivity Analysis

Performing a sensitivity analysis is valuable for assessing the impact of potential measurement errors on the parameters of a model, providing insights into the model's robustness. Results from previous extensive sensitivity analyses, particularly focusing on the effects of Hill-type muscle-tendon model parameters within musculoskeletal models [122, 114, 142, 40, 5, 31, 19, 28] indicate that the tendon slack length and maximal isometric force are the most sensitive parameters. Therefore, in this study, we conducted a sensitivity analysis on our proposed branching modeling method, focusing on these two parameters, to investigate their influence on the joint kinematics of the toy musculoskeletal models introduced in the previous section. Because all the tendons in these toy musculoskeletal systems were represented by ligaments, the sensitivity analysis on the tendon slack length should be equivalent with a sensitivity analysis on the ligament resting length in this context.

To assess the influence of changes in maximal isometric force, we applied ten perturbations to their nominal values, varying from -10% to $+10\%$ in 2% increments. Initially, we planned to use this same range for the nominal ligament resting length in each toy model. However, in most toy musculoskeletal systems, tendons are considerably longer compared to muscle length. Consequently, substantial perturbations, particularly negative ones, to ligament resting length could lead to simulation failures in Opensim. This issue arises because negative perturbations cause the ligament to contract at the start of the simulation when the muscle is not fully activated. The larger the negative perturbation, the more the ligament contracts. If such contraction moves the extra body, representing tendon branching points outside the two consecutive via points at the ends of the cylinder body, the simulation halts. As a result, we reduced the perturbation range for ligament resting length to -5% to $+5\%$, also in 2% steps. Notably, even this altered range is large due to the tendons' considerable length in these systems. While it was possible to address simulation crashes by manually adjusting parameters in the toy models, we avoided this due to the numerous parameters, their complex interrelations, and their complex effects on the resulting motions, especially in the context of branching. Therefore, we chose to simply reduce the perturbation range.

It is important to note that as there was more than one tendon in all of these toy musculoskeletal systems; we have chosen to apply the perturbation of the ligament resting length to all the ligaments simultaneously. For each perturbation, a new simulation was run under the same excitation signal used for the original experiments. Sensitivity analyses were conducted on both the conventional method and the proposed method.

To quantify the effect of the perturbation on the resulting motion, we re-applied the metric introduced:

$$\Delta = \max_t \sum_{i=1}^{n} \frac{\left|\tilde{\theta}_i^{pro}(t) - \theta_i^{pro}(t)\right|}{\bar{\theta}_i} \times 100\%, \qquad t \in [0, T] \tag{3.2}$$

This time, instead of comparing the proposed method with the conventional method, we compared the perturbed with unperturbed model for both the proposed method and the conventional branching modeling methods. Here, $\tilde{\theta}_i^{pro}(t)$ is the perturbed joint angle of i-th joint at time t. $\theta_i^{pro}(t)$ is the unperturbed joint angle of i-th joint at time t.

3.3 Results and Discussion

3.3.1 Model Construction

To demonstrate the use of the designed software for building articulated musculoskeletal models of highly jointed systems, the toy musculoskeletal systems introduced in the previous section were constructed by running through the entire software model construction pipeline. In addition, joint coordinate reporters and muscle controllers are assigned to each model using the OpenSim API. This setup allows for the recording and post-analysis of simulated motion in response to predefined muscle excitation signals.

The CSV data files and code for this model construction and the associated simulation results are available at https://github.com/EMBiRLab/ArborSim.

3.3.2 Comparative Simulations

The joint kinematics of the musculoskeletal models, which represent articulated systems built using the conventional and proposed methods, showed large differences. The quantitative differences between the derived motions, evaluated using the metric (3.1), are presented in Fig 3.5.

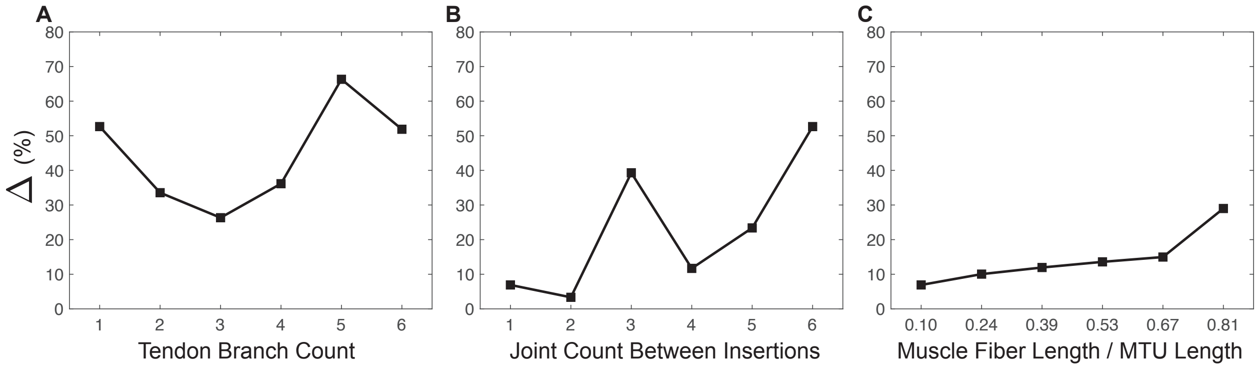

Figure 3.5: **Quantitative motion comparison.** Quantitative differences between the proposed branching modeling method and the conventional branching modeling method, evaluated by the metric (3.1), on the resulting motions of articulated systems with a varied number of tendon branches (**A**), a varied number of joints between insertions (**B**), and varied muscle fiber length to the MTU length ratios (**C**) are shown, respectively.

Specifically, within the category of different numbers of tendon branches (Fig 3.5A), the range of Δ across six simulations was [26.33%, 66.31%]. The peak value of this difference occurred when there were five tendon branches. Overall, as the number of tendon branches increased, the difference in joint kinematics first decreased and then increased. This trend suggested a complex, nonlinear relationship between in branched tendon systems.

Within the category of different numbers of joints between insertions (Fig 3.5B), the range of Δ across six simulations was [3.34%, 52.65%]. The largest Δ occurred when the branched tendons in the models spanned six joints between insertions. Disregarding the abrupt increase in the percentage change at three joint counts between insertions, a general incremental trend was observed in the motion differences with respect to the joint count between insertions. However, this jump might once again indicate a more complex, possibly nonlinear relationship.

Within the category of different muscle fiber length to MTU length ratios (Fig 3.5C), the range of Δ across six simulations was [6.91%, 28.99%]. The largest Δ occurred when the muscle spanned five joints, with the corresponding ratio being 0.81. There was a consistent trend of increasing motion differences as the muscle fiber length to MTU length ratio increased. However, compared to the other two categories, variations in the muscle fiber length to MTU length ratios resulted in relatively smaller changes and variance in Δ values.

To visually demonstrate the motion differences between the two modeling methods, we included a figure that highlights the comparison with the largest kinematic differences, determined by the metric (3.1), across each category (Fig 3.6).

Collectively, these results indicate that the conventional and proposed branching modeling methods exhibit notable differences in predicting joint kinematics. In particular, these differences become more pronounced as the complexity of muscle-tendon branching architectures increases, which involves numerous tendon branches, and an extensive number of joints spanned by branched tendon insertions. The resulting large differences can be concerning in fields demanding precise movement prediction, such as surgery outcomes, especially as the complexity of the muscle-tendon networks escalates in systems like human hands, spines, and elongated tails. The effect is likely most pronounced in elongated tails due to the absence of additional anatomical structures that limit movement in spines and hands. The results here suggest the pivotal role of the physical interactions between branched muscles and tendons in highly jointed systems. The study, though preliminary and focused on a kinematics comparison in toy musculoskeletal systems, highlights the necessity for continued in-depth exploration and advancement in the understanding of complex muscle-tendon branching phenomena and the branching modeling techniques in future works.

3.3.3 Sensitivity Analysis

The resulting deviations from the nominal movement, evaluated by (3.2), showed large variability across different categorized muscle-tendon branching structures. The effects of the perturbations on maximal isometric force, and ligament resting length on the joint kinematics are shown in Figs 3.7 and 3.8, respectively.

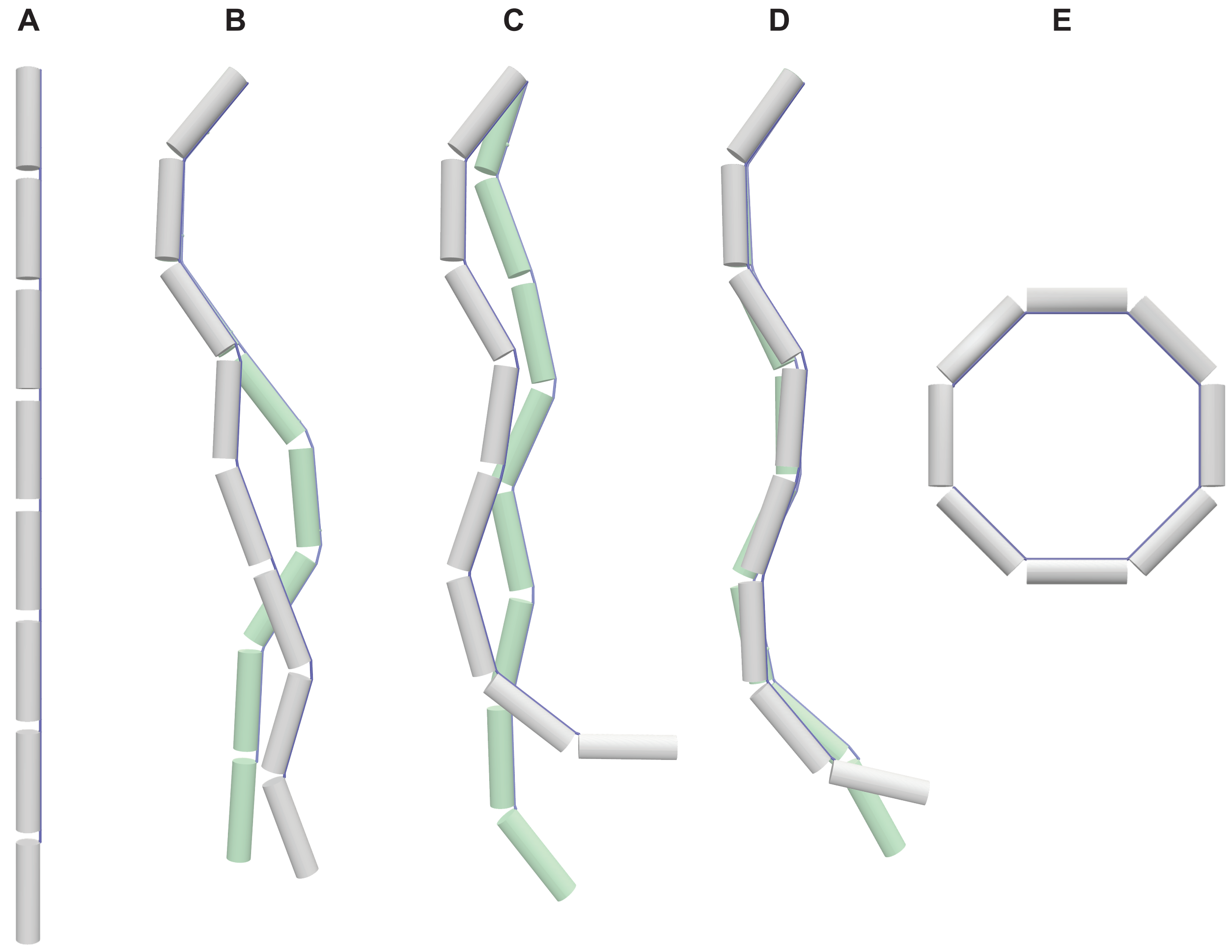

Figure 3.6: **Qualitative motion comparison.** (**A**) displays the zero-pose of the models. For (**B**), (**C**), and (**D**), each panel presents two comparative snapshots from the simulation that exhibited the highest Δ out of six simulations per category. These snapshots represent the largest motion differences between the conventional branching modeling method (depicted in white grey) and the proposed branching modeling method (depicted in light green). Specifically, (**B**) compares models with varying numbers of tendon branches, (**C**) compares models with different joint counts between insertions, and (**D**) comprares models with varied muscle fiber to MTU length ratios. (**E**) illustrates the pose where each joint is set to its maximum value within the range of motion.

Regardless of the muscle-tendon branching structure category, the models' joint kinematics was more sensitive to ligament resting length than to muscle maximal isometric force. This is consistent with previous studies [5, 40, 114, 122, 31, 142], emphasizing the crucial role of the tendon slack length plays in the joint kinematics of the musculoskeletal systems.

Regarding the sensitivity to muscle maximal isometric force, specifically within the categories of varying numbers of joints between insertions (Fig 3.7B) and varying muscle fiber length to MTU length ratios (Fig 3.7C), the observed Δ values were consistently low and stable. This suggests a low impact of these categories and variations within these categories on joint movement sensitivity. Within the category of varying numbers of tendon branches (Fig 3.7A), systems with fewer than 4 tendon branches exhibited a similar low sensitivity. However, for systems with 4 or more tendon branches, the joint kinematics showed a notably increased sensitivity for the proposed branching modeling methods. Moreover, the models built using the proposed method were significantly more sensitive than those built using the conventional method.

Regarding the sensitivity to tendon/ligament resting length, within the category of varying numbers of tendon branches (Fig 3.8A), the proposed method yielded higher Δ values compared to the conventional method, suggesting that the proposed method is more responsive to changes in number of tendon branches. In the category of varying numbers of joints between insertions (Fig 3.8B), both methods demonstrated similar levels of Δ values, indicating a comparable sensitivity to the number of joints between insertions. In the category of varying muscle fiber length to MTU length ratios (Fig 3.8C), an increase in the ratio of muscle fiber length to MTU length resulted in a decrease in Δ values for the conventional method. Conversely, the proposed method exhibited a relativel consistent level of Δ values across different ratios. Additionally, at each ratio, the proposed method showed higher Δ values than the conventional method, highlighting its higher sensitivity to the muscle fiber length to MTU length ratio.

These findings collectively underscore the importance of considering tendon slack length and the complexity of tendon branching when modeling muscle-tendon systems. They also highlight the enhanced responsiveness of the proposed method to key biomechanical parameters, suggesting its potential for facilitating more realistic simulations in biomechanics research.

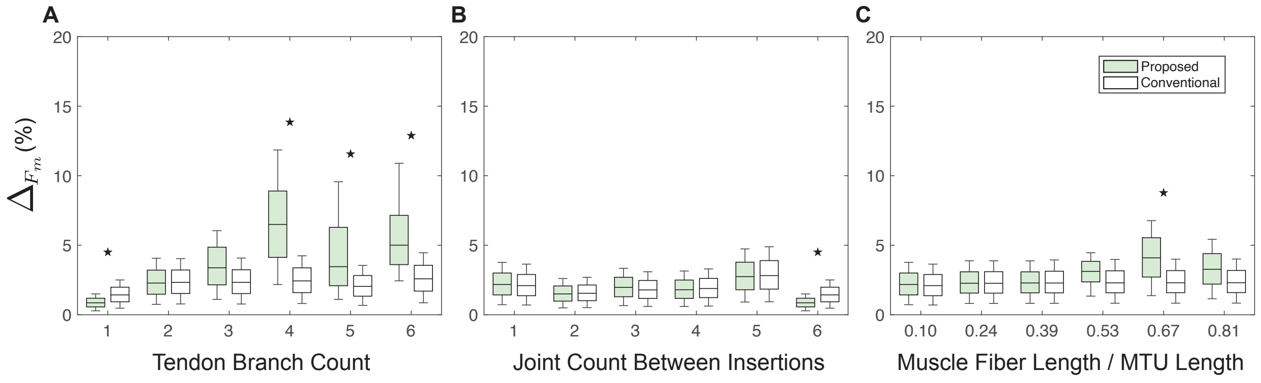

Figure 3.7: **Maximal isometric force sensitivity analysis.** Impact of maximal isometric force changes in the proposed versus conventional method on articulated system movement, measured by metric (3.2) across variations in tendon branch count (**A**), joint count between insertions (**B**), and muscle fiber length to the MTU length ratio (**C**). Solid black stars, if present, above the box plots indicate that the difference in Δ values between two groups, modeled by the conventional and proposed methods, is significant (P-value < 0.05).

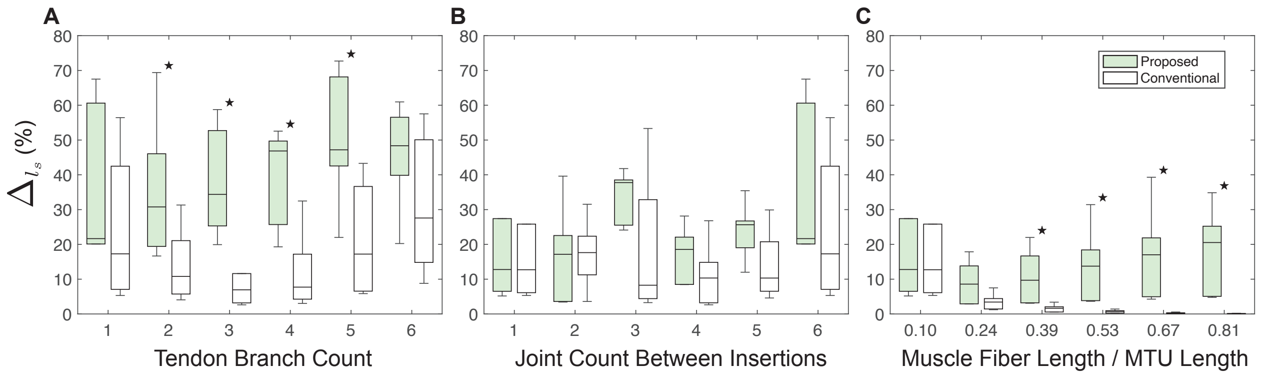

Figure 3.8: **Ligament resting length sensitivity analysis.** Impact of ligament resting length changes in the proposed versus conventional method on articulated system movement, measured by (3.2) across variations in tendon branch count (**A**), joint count between insertions (**B**), and muscle fiber length to the MTU length ratio (**C**). Solid black stars, if present, above the box plots indicate that the difference in Δ values between two groups, modeled by the conventional and proposed methods, is significant (P-value < 0.05).

3.3.4 Computational Time

In the proposed branching modeling method, introducing extra bodies of extremely small mass could potentially extend computational time due to potentially stiffer dynamics. To evaluate computational performance, we compared the computational times of the simulations using the proposed branching modeling method with those using the conventional branching modeling method. The comparison results are documented in Table 3.1.

In the table, for each muscle-tendon architecture category, we considered all the simulations presented in Figs 3.5, 3.7, and 3.8 to calculate the average computational time required to complete a simulation, each having a duration of $T = 3$ (sec).

Overall, the average computational times were similar between the two branching modeling methods. In the category of varying numbers of tendon branches, the simulations using the proposed methods took longer than those using the conventional method. This likely suggests that the more extra bodies of extremely small mass were introduced to the systems, the simulation time increased. It is important to note that the effect of these extra bodies on computational time is influenced by their mass. Here, the mass of the extra bodies used was 5×10^{-4} kg, which was 0.5% of a single vertebra (i.e., a single cylinder) in the models. Increasing the mass to a larger value, while still keeping it reasonably small relative to the body mass, can mitigate the slowdown effect.

Table 3.1: Computational time comparison

Architecture Category	**Conventional (sec)**	**Proposed (sec)**
Tendon Branch Count	4.44	4.92
Joint Count Between Insertions	4.06	4.29
Muscle Fiber Length / MTU Length	4.06	3.57

3.3.5 Future Tool Development

While the current version of the tool has proven effective by going through the procedure to create the toy models, we recognize opportunities for advancement and refinement. Several functionalities and improvements can be envisioned to extend its applicability and ease of use in the future. For example, in the current implementation of the "**Transformer**", global landmarks are required to establish joint coordinate systems, representing the axes of these systems. Identifying landmarks that can depict the axes of coordinate systems is indeed a crucial goal. However, acquiring these landmark points often necessitates identifying other biological references as a preliminary step. At present, we assume that this process is completed prior to using the designed tool. Nevertheless, integrating this preliminary step into the tool could further streamline the building process.

Also, the current implementation of the "**Brancher**" requires users to input anatomical and physiological parameters. Automatic parameter allocation could be realized by incorporating some predefined assumptions in future iterations of the software. Besides these, other elements such as complex coupled joints in which the spatial transformation might be dependent on other joint coordinates may also be included in future iterations of the tool to represent more complex joint movements observed in biological systems.

3.4 Conclusions

In this study, we introduced an extensible open-source tool, built upon OpenSim API, for constructing detailed, articulated musculoskeletal models of highly jointed biological systems. The tool was designed to facilitate the generation of musculoskeletal models from scratch using just CT scan and dissection data. Additionally, we introduced a method for modeling branched muscles and tendons that considers the interactions between them. This proposed method was implemented and integrated into the tool, alongside the conventional branching modeling method, providing users with a choice between two options of modeling branched muscles and tendons.

Using this tool, we created toy musculoskeletal systems featuring diverse muscle and tendon branching architectures, with a focus on branched tendons. We compared the proposed branching modeling method with the conventional method by conducting comparative simulations on these toy models. Our results showed large differences in the resulting movement between the models developed using the two distinct branching modeling methods. We believe the findings will persuade researchers of the necessity to delve into branching modeling in future studies. This exploration holds potential interest in several aspects. For instance, by incorporating explicit branching, researchers can examine potential actuation coupling mechanism inherent in branched muscles and tendons. Furthermore, exploring how different sets of branches coordinate could offer insights into whether there are simple control strategies for managing complex behaviors.

With the tool now introduced, in future studies, we aim to collect necessary material data of real muscle-tendon architectures to validate and refine the proposed branching method. We hope that our work will empower researchers to construct detailed, articulated musculoskeletal models of various systems, particularly those with multiple joints and complex muscle-tendon architectures, and study their contributions to animal motion.

CHAPTER 4

Linear Koopman Realizations for Data-driven Modeling and Control of Complex Dynamics

4.1 Introduction

Continuum appendages exhibit highly complex nonlinear dynamics due to their inherent composition, featuring a substantial number of articulated joints. Additionally, the presence of soft tissues introduces even more nonlinearities. These nonlinearities significantly raise the computational cost of analyzing and controlling these models, especially when optimizations are necessary. Therefore, the ability to accurately represent the dynamics of such complex systems while simultaneously enhancing their accessibility for analysis and control becomes exceedingly valuable.

Deep learning in neural networks has gained increasing popularities because of their ability to acquire accurate models of complex systems. For example, in [120], researchers applied deep reinforcement learning on a continuum robot to achieve open-loop position control. However, the structure of these types of models is not amenable to existing model-based control design techniques, since they are typically nonlinear and may not be easily invertible. Also, training a neural network model amounts to solving a non-convex nonlinear optimization problem in which the global convergence may not be guaranteed.

An alternative to deep learning and artificial neural networks is a data-driven modeling and control approach based on Koopman operator theory. In contrast to deep learning and artificial neural networks, this approach yields a linear model representation of a nonlinear controlled dynamical system from input-output data by leveraging the linear structure of the Koopman operator [25, 84, 85, 26, 108], and controlling them using established linear control methods [2, 69, 4, 82]

In theory, this approach involves *lifting* the state-space to an infinite-dimensional space of scalar functions (referred to as observables), where the flow of such observables along trajectories of the nonlinear dynamical system is described by the *linear* Koopman operator. In practice, however, it is infeasible to compute an infinite-dimensional operator, so a process called Extended Dynamic Mode Decomposition (EDMD) [138] is typically employed to compute a finite-dimensional projection of the Koopman operator onto a finite-dimensional subspace of all observables (scalar functions). This approximation of the Koopman operator describes the evolution of the output variables themselves, provided that they lie within the finite subspace of observables upon which the operator is projected. Hence, this approach makes it possible to control the output of a nonlinear dynamical system using a controller designed for its linear Koopman representation.

In this chapter, we introduce a Koopman-based modeling and control framework. We demonstrate the efficacy of this framework by applying it to a type of continuum robotic system, also known as soft robots. Since these soft continuum robots are made of intrinsically soft and/or compliant materials, they can deform continuously along their bodies, thereby possessing infinite degrees of freedom in theory. Thus, they exhibit a level of dynamic complexity comparable to, or even exceeding, that of biological continuum appendages. This makes them an excellent candidate for demonstrating the effectiveness of the Koopman-based modeling and control framework in modeling and controlling complex systems. We demonstrate that the Koopman-based data-driven modeling and control method creates dynamic models that capture the system's true behavior more accurately than the models generated by several other state-of-the art 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. Building upon the linear Koopman models, we develop model predictive control algorithms and illustrate their effectivness in controlling these real soft robotic systems.

The rest of this chapter is organized as follows: In Section 4.2, we formally introduce the Koopman operator and describe how it is used to construct models of nonlinear dynamical systems from data. In Section 4.3, we describe how the Koopman model can be used to construct a linear model predictive controller (MPC) and a nonlinear model predictive controller (NMPC), with the NMPC controller serving for comparative purposes. In Section 4.4, we demonstrate the application of our methodology to a real soft continuum robotic system. We compare the performance of the model identified through the Koopman-based modeling method with that of several models generated using state-of-the-art nonlinear system identification techniques, and evaluate the performance of Koopman-based model predictive controllers. In Section 4.5, we extend the Koopman-based modeling and control methodology to include external forces encountered by continuum appendages during interactions with objects or the environment, which can be frequent during the system's movement. Specifically, we describe how to incorporate external loading conditions into the linear Koopman realizations for robust modeling and control under variable loading conditions. In Section 4.6, we demonstrate that efficacy of the extension of the framework on a continuum robotic system.

4.2 Koopman-based System Identification

Any finite-dimensional, Lipschitz continuous nonlinear dynamical system has an equivalent infinite-dimensional linear representation in the space of all scalar-valued functions of the system's state [71, Definition 3.3.1]. This linear representation, which is called the Koopman operator, describes the flow of functions along trajectories of the system. While it is not possible to numerically represent the infinite-dimensional Koopman operator, a projection of the Koopman operator onto a finite-dimensional subspace can be employed.

This section describes a method for approximating the Koopman operator to construct linear models of nonlinear controlled discrete dynamical systems from input/state data. Particularly, Section 4.2.1 introduces the Koopman operator. Section 4.2.2 demonstrates how to identify an approximation of the Koopman operator as a matrix. Section 4.2.3 describes how to construct a linear model from a matrix approximation of the Koopman operator.

4.2.1 The Koopman Operator

Consider a dynamical system governed by the following differential equation

$$\dot{x}(t) = F(x(t), u(t)) \tag{4.1}$$

where $x(t) \in X \subset \mathbb{R}^n$ is the state of the system and $u(t) \in U \subset \mathbb{R}^m$ is the input of the system at time $t \geq 0$, X, U are compact subsets, and F is a continuously differentiable function. Denote by $\phi_\tau : X \times U \to X \times U$ the *flow map*, where $\phi_\tau(x(0), u(0)) = (x(\tau), u(0))$ and $x(\tau)$ is the solution to (5.1) at time τ when beginning with the initial condition $x(0)$ 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 continuous real-valued functions with compact domain $X \times U \subset \mathbb{R}^n \times \mathbb{R}^m$. Elements of $\mathcal{F}$ are called *observables*. In $\mathcal{F}$, the flow of the system is characterized by the set of Koopman operators $\mathcal{K}_t : \mathcal{F} \to \mathcal{F}$, for each $t \geq 0$, which describes the evolution of the observables $f \in \mathcal{F}$ along the trajectories of the system according to the following definition:

$$\mathcal{K}_t f = f \circ \phi_t, \tag{4.2}$$

where $\circ$ indicates function composition. $\mathcal{K}_t$ is a linear operator even if the system (5.1) is nonlinear, since for $f_1, f_2 \in \mathcal{F}$ and $\lambda_1, \lambda_2 \in \mathbb{R}$

$$\begin{aligned} \mathcal{K}_t(\lambda_1 f_1 + \lambda_2 f_2) &= \lambda_1 f_1 \circ \phi_t + \lambda_2 f_2 \circ \phi_t \\ &= \lambda_1 \mathcal{K}_t f_1 + \lambda_2 \mathcal{K}_t f_2. \end{aligned} \tag{4.3}$$

Specifically, the Koopman operator $\mathcal{K}_\tau$ advances the value of an observable forward along the trajectories of the underlying dynamical system as

$$\mathcal{K}_\tau f(x(t), \tilde{u}) = f(x(t+\tau), \tilde{u}) \tag{4.4}$$

where $\tilde{u}$ is a constant input over the interval $[t, t+\tau]$.

Note that this is true for *any* observable function f, so the Koopman operator can also be used to advance the system state itself by applying it to the set of functions $\{f_i : f_i(x(t), \tilde{u}) = x_i(t)\}_{i=1}^n$, advancing their values according to (4.4), and stacking the results as a vector:

$$x(t+\tau) = \begin{bmatrix} \mathcal{K}_\tau f_1\left(x(t), \tilde{u}\right) & \cdots & \mathcal{K}_\tau f_n\left(x(t), \tilde{u}\right) \end{bmatrix}^\top . \tag{4.5}$$

In this way, the Koopman operator provides an infinite-dimensional linear representation of a nonlinear dynamical system [27].

4.2.2 Identification of the Koopman Operator

While the Koopman operator is theoretcially infinite-dimensional, for pratical uses, using the Extended Dynamic Mode Decomposition (EDMD) algorithm [138, 85], we identify a finite-dimensional matrix approximation of the Koopman operator via linear regression.

The remainder of this subsection describes the mathematical foundation 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 $\{\psi_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}\}_{i=1}^N$ (e.g. monomials, sinusoids), and define the *lifting function* $\psi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^N$ as a vector containing all the basis functions:

$$\psi(x, u) := \begin{bmatrix} \psi_1(x, u) & \cdots & \psi_N(x, u) \end{bmatrix}^\top . \tag{4.6}$$

Then any observable $\bar{f} \in \bar{\mathcal{F}}$ can be expressed as a linear combination of the basis functions

$$\bar{f} = \theta_1 \psi_1 + \cdots + \theta_N \psi_N \tag{4.7}$$

where each $\theta_i \in \mathbb{R}$ is the weight associated with each basis function. To aid in presentation, define the vector of coefficients $\theta := [\theta_1 \, \cdots \, \theta_N]^\top$. By (4.7), $\bar{f}$ evaluated at x, u can be expressed as

$$\bar{f}(x,u) = \theta^\top \psi(x,u) \tag{4.8}$$

where θ 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:

$$\bar{\mathcal{K}}_\tau \theta = \theta' \tag{4.9}$$

where θ, θ' are each vector representations of observables in $\bar{\mathcal{F}}$.

Since $\bar{\mathcal{K}}_\tau$ is desired to carry the characteristic of $\mathcal{K}_\tau$'s operation on the observables, by (4.2) the following should be true for all $x \in X$ and $u \in U$

$$\bar{\mathcal{K}}_\tau \bar{f}(x,u) = \bar{f} \circ \phi_\tau(x,u) \tag{4.10}$$

Substituting (4.8), we have

$$(\bar{\mathcal{K}}_\tau \theta)^\top \psi(x,u) = \theta^\top \psi \circ \phi_\tau(x,u) \tag{4.11}$$

$$\bar{\mathcal{K}}_\tau^\top \psi(x,u) = \psi \circ \phi_\tau(x,u) \tag{4.12}$$

where (4.12) follows by cancelling θ in (4.11) on both sides. Notice that $\bar{\mathcal{K}}_\tau$ acts as a linear flow map on the lifting function ψ. Hence, it follows that for a given $x \in X$ and $u \in U$, solving (4.12) for $\bar{\mathcal{K}}_\tau$ yields the best approximation of $\mathcal{K}_\tau$ on $\bar{\mathcal{F}}$ in the L^2-norm sense [104]:

$$\bar{\mathcal{K}}_\tau = \left(\psi(x,u)^\top\right)^\dagger \left(\psi \circ \phi_\tau(x,u)\right)^\top \tag{4.13}$$

where superscript † denotes the Moore-Penrose pseudoinverse of a matrix.

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

$$a[v] := x[v] \tag{4.14}$$

$$b[v] := \phi_{T_s}(x[v], u[v]), \tag{4.15}$$

$x[v]$ denotes the state corresponding to the v^{th} measurement, $u[v]$ is the constant input applied between $a[v]$ and $b[v]$, and T_s is the sampling period, which is assumed to be identical for all snapshots. We then lift all of the snapshots according to (4.6) and compile them into the following $V \times N$ matrices:

$$\Psi_a := \begin{bmatrix} \psi(a[1], u[1])^\top \\ \vdots \\ \psi(a[V], u[K])^\top \end{bmatrix}, \qquad \Psi_b := \begin{bmatrix} \psi(b[1], u[1])^\top \\ \vdots \\ \psi(b[V], u[V])^\top \end{bmatrix}. \tag{4.16}$$

Following from (4.13), $\bar{\mathcal{K}}_{T_s}$ is obtained as the best linear least-square estimate of Ψ_b given Ψ_a, which is the solution to

$$\min_{\bar{\mathcal{K}}_{T_s}} \left\| \Psi_a \bar{\mathcal{K}}_{T_s} - \Psi_b \right\|_2^2 . \tag{4.17}$$

The analytical solution to (4.17) is

$$\bar{\mathcal{K}}_{T_s} := \Psi_a^\dagger \Psi_b, \tag{4.18}$$

where † denotes the Moore-Penrose psuedoinverse.

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

To incorporate one or more delays, we define the snapshot pairs as

$$a[v] = \begin{bmatrix} x[v]^\top, & x[v-1]^\top & \dots & x[v-d]^\top, & u[v-1] & \dots & u[v-d] \end{bmatrix}^\top \tag{4.19}$$

$$b[v] = \begin{bmatrix} x[v+1]^\top, & x[v]^\top & \dots & x[v-d+1]^\top, & u[v-1] & \dots & u[v-d] \end{bmatrix}^\top \tag{4.20}$$

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

4.2.3 Linear Model Realization Based on the Koopman Operator

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

$$\begin{aligned} z[j+1] &= Az[j] + Bu[j] \\ x[j] &= Cz[j] \end{aligned} \tag{4.21}$$

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

The goal is to construct $\bar{\mathcal{K}}_{T_s}$ such that it is decomposable into a linear system representation like (4.21). This can be achieved by a suitable choice of basis functions $\{\psi_i\}_{i=1}^N$.

Define the first $N - m$ basis functions as functions of the state only, and the last m basis functions as indicator functions on each component of the input,

$$\psi_i(x, u) = g_i(x), \quad \forall i \in \{1, \ldots, N - m\} \tag{4.22}$$

$$\psi_i(x, u) = u_i, \quad \forall i \in \{i = N - m + 1, \ldots, N\} \tag{4.23}$$

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

$$g(x) := \begin{bmatrix} g_1(x) & \cdots & g_{N-m}(x) \end{bmatrix}^\top . \tag{4.24}$$

Note by (4.17), $\bar{\mathcal{K}}_{T_s}^\top$ is the best approximation of a transition matrix between the lifted snapshots in the L^2-norm sense, i.e., given the lifting functions defined in (4.22) and (4.23), $\bar{\mathcal{K}}_{T_s}$ is the minimizer to

$$\min_{\check{\mathcal{K}}} \sum_{v=1}^{V} \left\| \check{\mathcal{K}}^\top \begin{bmatrix} g(a[v]) \\ u[v] \end{bmatrix} - \begin{bmatrix} g(b[v]) \\ u[v] \end{bmatrix} \right\|_2^2 \tag{4.25}$$

and the best realizations of A and B we desire minimize

$$\min_{\check{A}, \check{B}} \sum_{v=1}^{V} \left\| \check{A} g(a[v]) + \check{B} u[v] - g(b[v]) \right\|_2^2 . \tag{4.26}$$

Therefore, comparing (4.25) and (4.26), the best A and B matrices are embedded in $\bar{\mathcal{K}}_{T_s}$ and can be isolated by partitioning it as follows:

$$\bar{\mathcal{K}}_{T_s}^\top = \begin{bmatrix} A_{(N-m)\times(N-m)} & B_{(N-m)\times m} \\ O_{m\times(N-m)} & I_{m\times m} \end{bmatrix} \tag{4.27}$$

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

We can also define the first n basis functions as indicator functions on each component of the state, i.e.

$$g_i(y) = y_i \quad \forall i \in \{1, \ldots, n\}. \tag{4.28}$$

Then, C matrix is defined

$$C = \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix}. \tag{4.29}$$

The process for constructing the proposed linear model is summarized in Algorithm 2.

```
Algorithm 2: Koopman Linear System Identification
    Input: . $\lambda$ , $\{a[v], b[v]\}$ and $u[v]$ for $v = 1, ..., V$
    Step 1: Lift data via (4.6)
    Step 2: Approximate Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (4.18)
    Step 3: Extract model matrices $A, B$ via (4.27)
    Output: $A, B, C := \begin{bmatrix} I_{n\times n} & O_{n\times(N-n)} \end{bmatrix}$
```

4.2.4 Nonlinear Model Realization Based on the Koopman Operator

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

$$\dot{x}(t) = F(x(t), u(t)). \tag{4.30}$$

Using the same lifting function (4.22) and (4.23) defined in the previous section, the Koopman operator is once again approximated using least-squares regression via (4.17).

To construct a model in the form of (4.30), we introduce the infinitesimal generator of the set of Koopman operators $G : \mathcal{F} \to \mathcal{F}$ [71, Equation 7.6.5], which is defined in terms of the vector field

F as

$$G = F \cdot \nabla_x. \tag{4.31}$$

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:

$$\mathcal{K}_t = e^{Gt} \tag{4.32}$$

In practice, our goal is to find a function $\bar{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that describes the behavior of the function F as accurately as possible in the L^2-norm sense on the finite dimensional subspace $\bar{\mathcal{F}}$. With the approximation of the Koopman operator $\bar{\mathcal{K}}_{T_s}$ in hand, we can solve for the infinitesimal generator $\bar{G}$ by inverting (4.32):

$$\bar{G} = \frac{1}{T_s} \log \bar{\mathcal{K}}_{T_s} \in \mathbb{R}^{N \times N} \tag{4.33}$$

where $\log$ denotes the principal matrix logarithm [58, 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 [84, Section III-B]. Therefore this method might fail if the number of data points is insufficient. In this instance, more system measurements can be taken to resolve the issue.

With $\bar{G}$ known, (4.31) can be used to identify $\bar{F}$. Consider G applied to an observable $f \in \mathcal{F}$. According to (4.31), this is equivalent to the inner product of the vector field F and the gradient of

f with respect to x:

$$Gf(x,u) = \frac{\partial f(x,u)}{\partial x} F(x,u). \tag{4.34}$$

Again using θ to denote the vector representation of an observable $\bar{f}$ and $a = [x^\top, u^\top]^\top$ to represent the augmented state with the input appended, the finite-dimensional equivalent of (4.34) is given by

$$(\bar{G}\theta)^T \psi(a) = \theta^T \frac{\partial \psi(a)}{\partial x} \bar{F}(x,u). \tag{4.35}$$

We seek the vector field $\bar{F}$ such that (4.35) holds as well as possible in the L^2-norm sense for all observed data. Therefore, we choose the least-squares solution to (4.35) over the set of all observed data points $\left\{a_v = [x_v^\top, u_v^\top]^\top | v = 1, ..., V\right\}$ which is given by

$$\bar{F}(x,u) = \begin{bmatrix} \frac{\partial \psi(a_1)}{\partial x} \\ \vdots \\ \frac{\partial \psi(a_V)}{\partial x} \end{bmatrix}^\dagger \begin{bmatrix} \bar{G}^T \\ \vdots \\ \bar{G}^T \end{bmatrix} \psi(a). \tag{4.36}$$

Algorithm 3 summarizes this nonlinear system identification process. For a more thorough treatment, see [84, 85].

Algorithm 3: Koopman Nonlinear System Identification

```
Input: . $\lambda$ , $\{a[v], b[v]\}$ and $u[v]$ for $v = 1, ..., V$
Step 1: Lift data via (4.6)
Step 2: Approximate Koopman operator $\bar{\mathcal{K}}_{T_s}$ via (4.18)
Step 3: Identify infinitesimal generator $\bar{G}$ by (4.35)
Step 4: Solve for vector field $\bar{F}$ via (4.36)
Output: . $\bar{F} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$
```

4.3 Model Predictive Control

A system model enables the design of model-based controllers that leverage model predictions to choose suitable control inputs for a given task. In particular, model-based controllers can anticipate future events, allowing them to optimally choose control inputs over a finite time horizon. A popular model-based control design technique is model predictive control (MPC), wherein one optimizes the control input over a finite time horizon, applies that input for a single time-step, and then optimizes again, repeatedly [113].

This section introduces several MPC controllers based on models constructed from the Koopman operator. Section 4.3.1 describes the MPC optimization problem for a linear Koopman model in the form of (4.21). Section 4.3.2 describes the MPC optimization problem for a nonlinear Koopman model in the form of (4.30). Section 4.3.3 discusses the relative computational efficiency of each approach.

4.3.1 Linear MPC

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

$$
\begin{aligned}
& \min_{\substack{\{u[i]\in\mathbb{R}^m\}_{i=0}^{N_h} \\ \{z[i]\in\mathbb{R}^N\}_{i=0}^{N_h}}} && z[N_h]^T G[N_h] z[N_h] + g[N_h]^T z[N_h] + \\
& && + \sum_{i=0}^{N_h-1} \Big(z[i]^T G[i] z[i] + u[i]^T H[i] u[i] + \\
& hangz && + g[i]^T z[i] + h[i]^T u[i] \Big) \qquad (4.37) \\
& \text{s.t.} && z[i+1] = \hat{A} z[i] + \hat{B} u[i], \quad \forall \{i\}_{i=0}^{N_h-1} \\
& && E[i] z[i] + F[i] u[i] \le b[i], \quad \forall \{i\}_{i=0}^{N_h-1} \\
& && z[0] = \psi(x[k])
\end{aligned}
$$

where $N_h \in \mathbb{N}$ is the prediction horizon, $G[i] \in \mathbb{R}^{N\times N}$ and $H[i] \in \mathbb{R}^{m\times m}$ are positive semidefinite matrices, and where each time the program is called, the predictions are initialized from the current lifted state $g(x[k])$. 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 (4.37) depend on the dimension of the lifted state N, [69] shows that these can be rendered independent of N by transforming the problem into its so-called "dense-form." Algorithm 5 summarizes the closed-loop operation of this Koopman-based linear MPC controller.

Algorithm 4: Koopman-based Linear MPC (K-MPC)

```
Input:  Prediction horizon: $N_h$
        Cost matrices: $G_i, H_i, g_i, h_i$ for $i = 0, ..., N_h$
        Constraint matrices: $E_i, F_i, b_i$ for $i = 0, ..., N_h$
        Model matrices: $\hat{A}, \hat{B}$
for $k = 0, 1, 2, ...$ do
    Step 1: Set $z[0] = g(x[k])$
    Step 2: Solve (4.37) 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
```

4.3.2 Nonlinear MPC

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

$$\begin{aligned}
\min_{u \in L^2([t_0,t_f];\mathbb{R}^m)} \quad & \int_{t_0}^{t_f} \Big((x(t) - x_{\text{ref}}(t))^T Q \,(x(t) - x_{\text{ref}}(t)) + \\
& + u(t)^T R u(t) \Big)\, dt \\
\text{s.t.} \quad & \dot{x}(t) = F\,(x(t), u(t))\,, \quad \forall t \in [t_0, t_f] \\
& x(t_0) = x_0 \\
& L(x(t), u(t)) \leq 0_q, \quad \forall t \in [t_0, t_f]
\end{aligned} \tag{4.38}$$

where $L^2([t_0, t_f]; \mathbb{R}^m)$ is the space of square integrable functions from $[t_0, t_f]$ to $\mathbb{R}^m$, $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ are positive semidefinite matrices, $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$ defines state and input constraints, and 0_q is the vector of all 0's with q-elements. We solve this problem by applying a commercial optimal control problem solver - GPOPS-II [101].

Solving (4.38) quickly enough for real-time receding horizon control is often not possible. In practice, the optimal control inputs must be computed offline then applied to the system in open-loop. Without feedback, the controller is incapable of overcoming model error that causes performance to degrade as the time-horizon increases. To counteract this shortcoming, a nonlinear MPC controller can be supplemented with a linear feedback controller based on the local linearization of the nonlinear model about the reference trajectory (see [13, Section 5.4]). At each time step k, the optimal input $u^*[k]$ is then given by the sum of the open-loop control input computed by solving the aforementioned NMPC offline, denoted $\overline{u}[k]$, and a feedback term based on the local linearization that is computed online, denoted $\delta_u[k]$,

$$u^*[k] = \overline{u}[k] + \delta_u[k] \tag{4.39}$$

where $\delta_u[k]$ is chosen such that it drives the locally linearized dynamics to zero. In practice, we compute $\delta_u[k]$ at each timestep by solving a linear MPC program with structure similar to (4.37).

4.3.3 LMPC vs. NMPC

Convex optimization programs, i.e. those which have a convex cost function and convex feasible set, have global extrema and can be solved reliably and efficiently [21]. Non-convex optimization programs, on the other hand, may exhibit multiple local extrema and there is no computationally tractable technique for finding global solutions to generic problems. Typically non-convex optimization methods only find local solutions that depend on an initial guess. Thus, for real-time optimal control problems, convexity is desirable. A major advantage of lifting and identifying a linear system model using the Koopman operator is that the LMPC control problem for such models is convex.

The MPC optimization problem described in Section 4.3.1 is convex because it has a quadratic cost function and model dynamics that are described by linear constraints. Since it is convex, it has a unique globally optimal solution that can efficiently be identified without initialization for models with thousands of states and inputs [21, 102, 129].

The NMPC optimization problem described in Section 4.3.2 also has a quadratic cost function, but it is not convex because it has model dynamics that are described by nonlinear constraints. As a result, algorithms to solve this problem typically require initialization and can struggle to find globally optimal solutions [106]. Though techniques have been proposed to improve the speed of such algorithms [101, 56] or even globally solve NMPC problems without requiring initialization [149], these formulations still take several seconds per iteration, which make them too slow to be applied in most real-time control scenarios. This difference is highlighted in Section 5.4 via computation time comparisons between LMPC and NMPC controllers for a soft robot.

4.4 Application: Pneumatic Soft Robot Arm

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

4.4.1 Robot Description: Soft Arm with a Laser Pointer

We experimentally evaluated the Koopman control approach on a soft robot arm performing the task of drawing shapes with a laser pointer, similar to the handwriting task described in [66]. The robot is a suspended soft arm with a laser pointer attached to the end effector (see Fig. 4.1). 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 [132]) adhered to a central foam spine by latex rubber bands (see Fig. 4.1). 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

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

three pressure regulators. The state is two-dimensional and corresponds to the position of the laser dot with respect to the center of the board in Cartesian coordinates.

4.4.2 Characterization of Dynamic Uncertainty

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

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

$$u[k] = \begin{bmatrix} 6\sin(\frac{2\pi}{T}kT_s) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{T}{3}) + 3 \\ 6\sin(\frac{2\pi}{T}kT_s - \frac{2T}{3}) + 3 \end{bmatrix} \tag{4.40}$$

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. 4.2 the trajectories over 210 periods are superimposed along with the average over all trials.

The standard deviation of this distribution is 0.215 cm. This inherent uncertainty will limit the tracking performance of the system, independent of the employed controller.

4.4.3 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

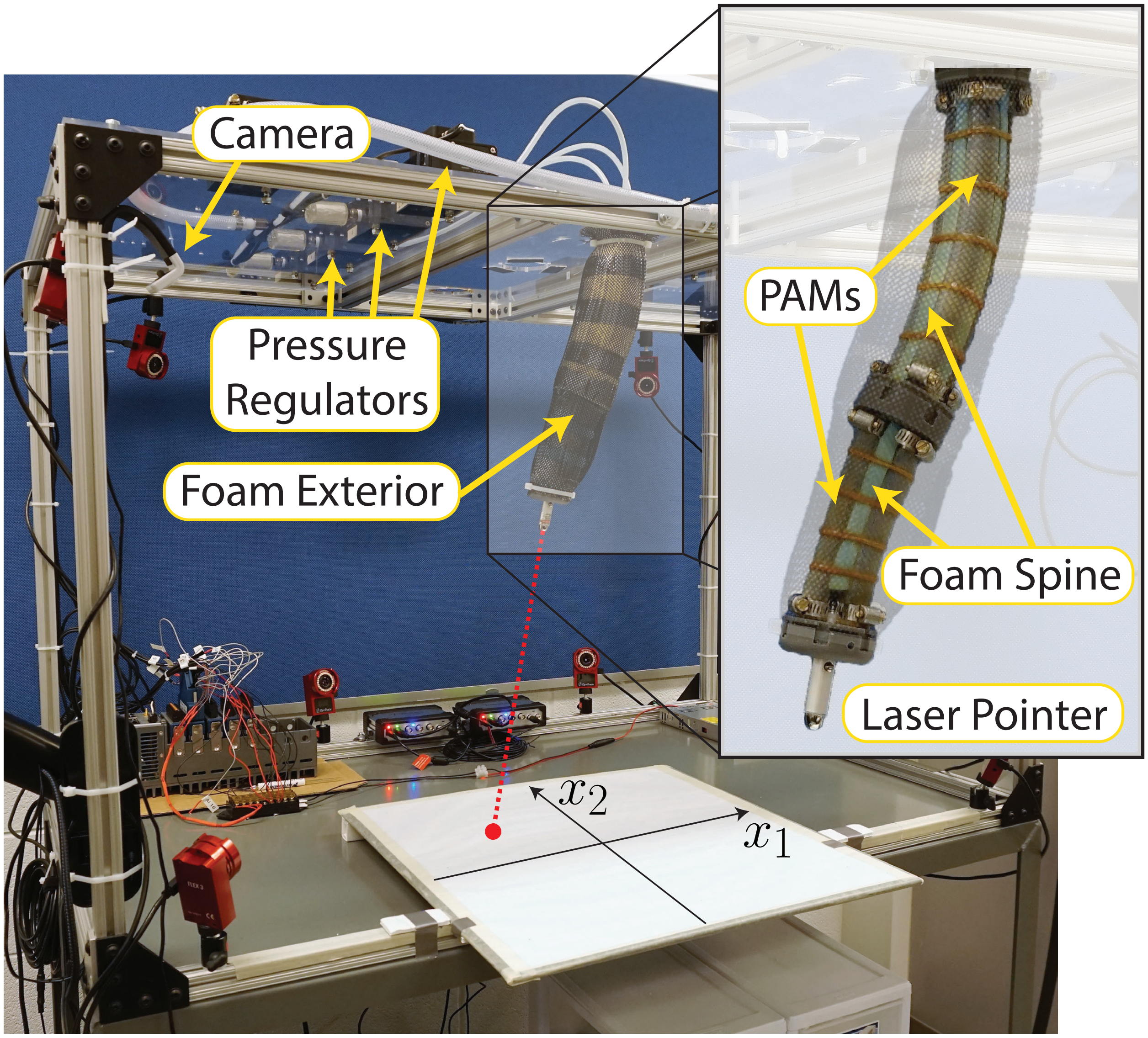

Figure 4.1: The soft robot consists of two bending segments encased in a foam exterior with a laser pointer attached to the end effector. A set of three pressure regulators is used to control the pressure inside of the pneumatic actuators (PAMs), and a camera is used to track the position of the laser dot.

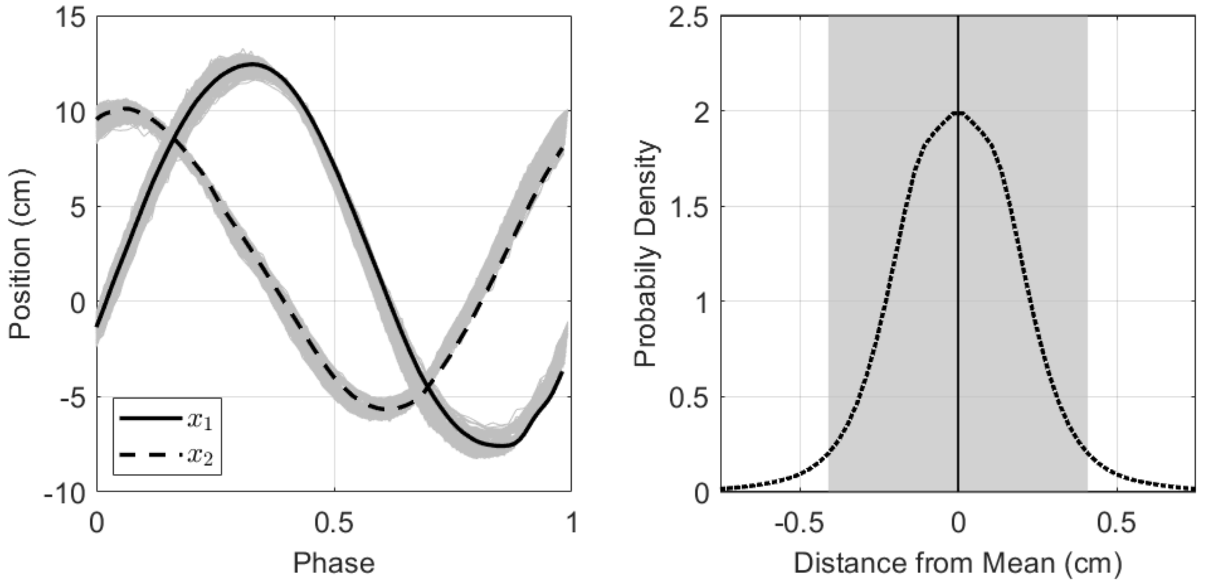

Figure 4.2: The left plot shows the average response of the soft robot system over a single period when the sinusoidal inputs of varying frequencies described by (4.40) are applied. All of the particular responses are subimposed in light grey. The right plot shows the distribution of trajectories about 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.

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

$$u_i[k] = \frac{(\Upsilon_{i,j+1} - \Upsilon_{i,j})}{T_u}\left(kT_s + \frac{(i-1)T_u}{3}\right) + \Upsilon_{i,j} \tag{4.41}$$

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

Four models were fit from data: a linear state-space model using the subspace method [136], a linear Koopman model using the approach from Section 4.2.3, a nonlinear Koopman model using the approach from Section 4.2.4, and a Nonlinear Autoregressive with External Input (NARX) neural network model identified using the Levenberg-Marquardt backpropogation algorithm. Each of these models was trained from the randomly generated data just once, independent of any specific task.

The linear state-space model provides a baseline for comparison and was identified from the same data as the Koopman models using the MATLAB System Identification Toolbox [83]. This model is a four dimensional linear state-space model expressed in observer canonical form.

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

$$a[k] = \begin{bmatrix} x[k]^\top & x[k-1]^\top & u[k-1]^\top \end{bmatrix}^\top \tag{4.42}$$

$$b[k] = \begin{bmatrix} \phi_{T_s}(x[k], u[k])^\top & x[k]^\top & u[k]^\top \end{bmatrix}^\top \tag{4.43}$$

and used an $N = 36$ dimensional set of basis functions consisting of all monomials of maximum degree 2. 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.

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

$$a[k] = \begin{bmatrix} x[k]^\top & u[k]^\top \end{bmatrix}^\top \tag{4.44}$$

$$b[k] = \begin{bmatrix} \phi_{T_s}(x[k], u[k])^\top & u[k]^\top \end{bmatrix}^\top , \tag{4.45}$$

and used an $N = 35$ dimensional set of basis functions consisting of all monomials of maximum degree 3.

The NARX neural network model was identified using the same set of $45,103$ snapshot pairs as the linear Koopman model, which incorporates a single delay $d = 1$. The model was trained using the MATLAB Neural Network Toolbox [83] with sigmoid activation functions and the number of hidden neurons was tuned from 5-20. The best results were achieved with 10 hidden neurons, so this is what was used for the model prediction comparison described in Section 4.4.4.

4.4.4 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 (4.40). 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. 4.3 shows the tracking performance for the case when the inputs have period $T = 6$ seconds. These results illustrate that both Koopman models generate more accurate predictions than the state space and neural network models with the nonlinear Koopman model producing slightly better predictions than the linear Koopman model.

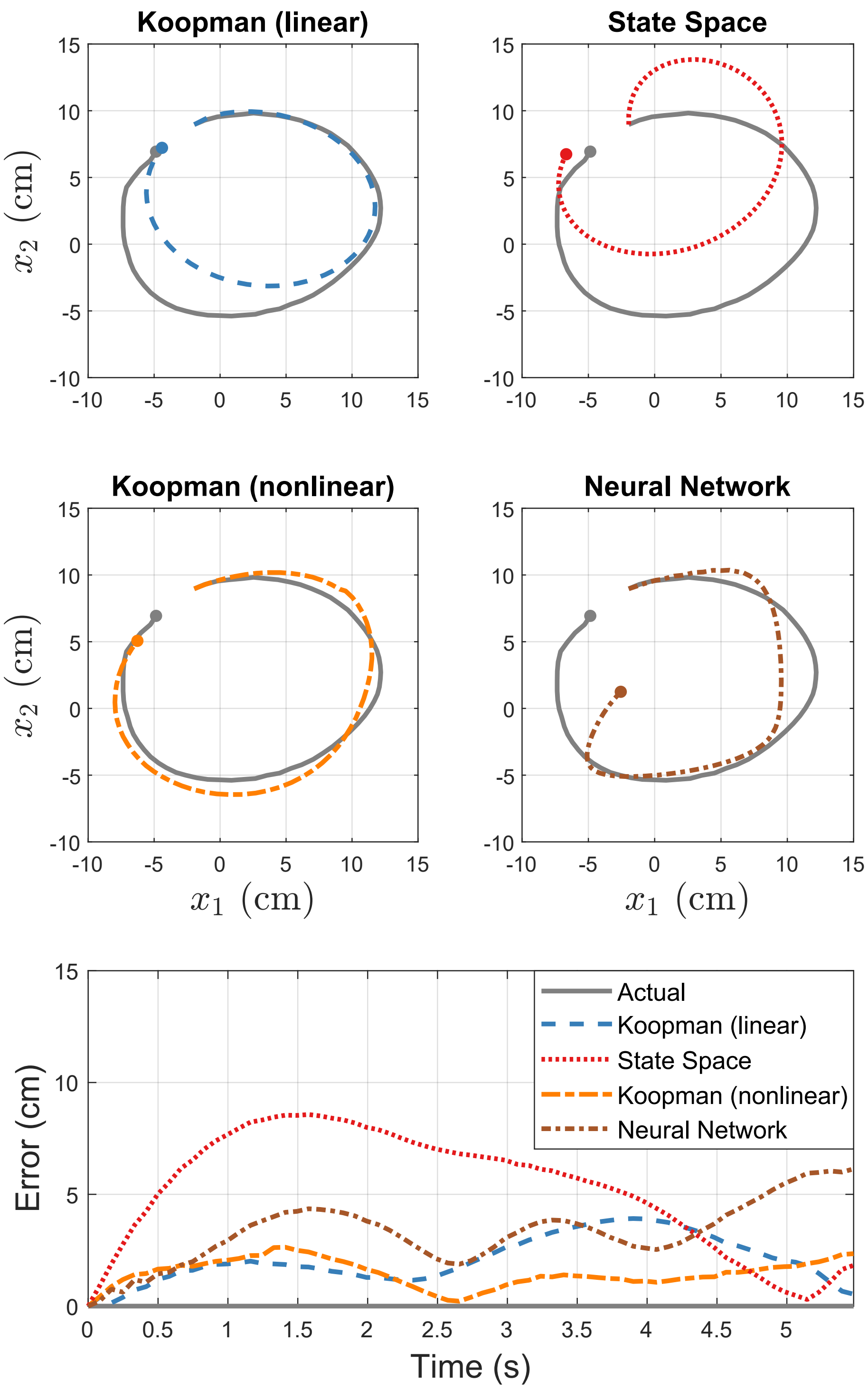

Figure 4.3: The actual response and the model predictions for the robot with the sinusoidal inputs described in (4.40) with period $T = 6$ seconds applied. The left plot shows the actual trajectory of the laser dot along with model predictions. The error displayed on the bottom plot is defined as the Euclidean distance between the predicted laser dot position and the actual position at each point in time.

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

Table 4.2: Average Euclidean Distance 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	0.95	0.96	0.99	0.96	0.02
K-NMPC-LL	0.43	0.37	0.39	0.40	0.03

4.4.5 Experiment 2: Model-Based Control Comparison

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

LMPC: 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,

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

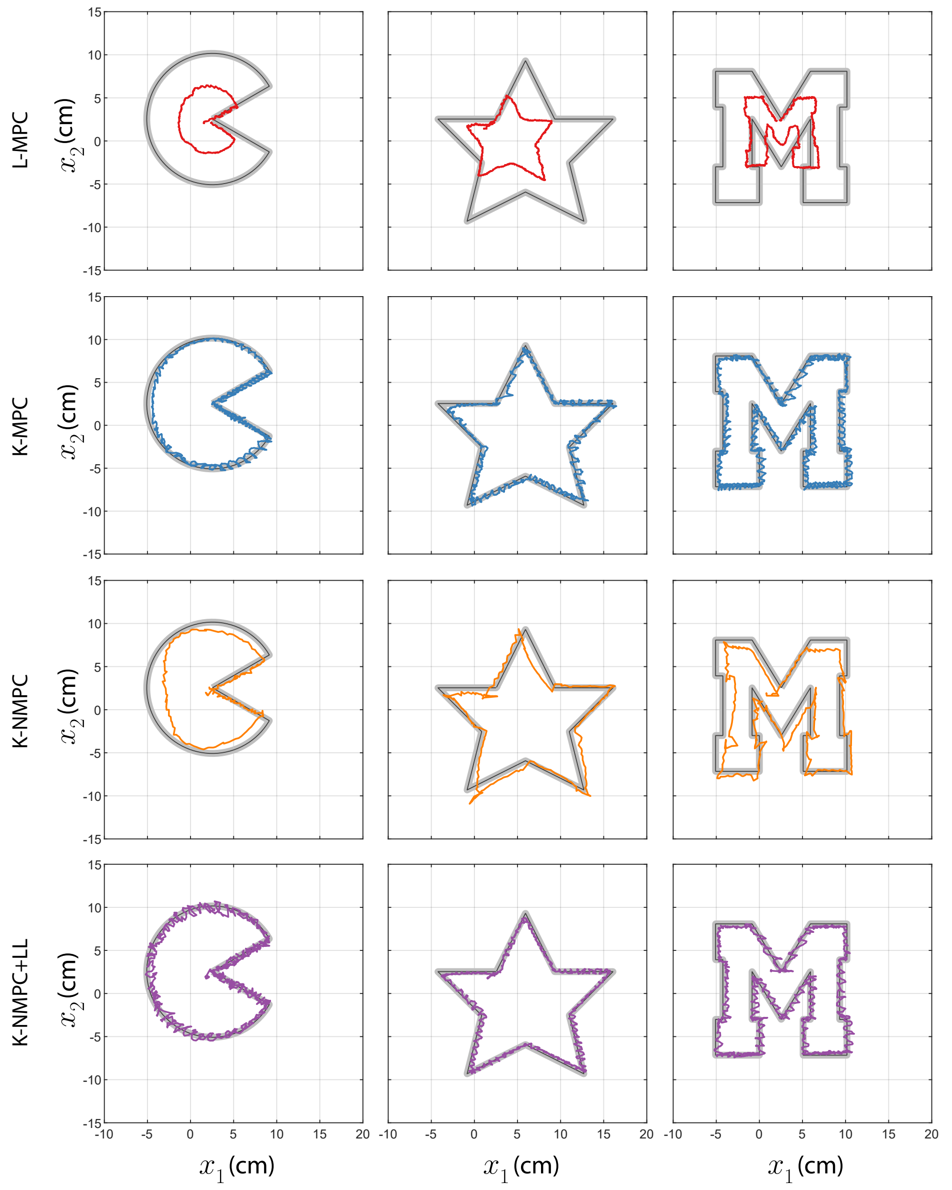

Figure 4.4: The results of the L-MPC controller (row 1, red), the K-MPC controller (row 2, blue), the K-NMPC controller (row 3, orange), and the K-NMPC+LL controller (row 4, purple) in performing trajectory following tasks 1-3. The reference trajectory for each task is subimposed in black as well as a grey buffer with width equal to two standard deviations of the noise probability density shown in Fig. 4.2.

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 (LMPC, K-MPC) both solve an optimization problem in the form of (4.37) at each time step using the Gurobi Optimization software [54]. 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 $y^{\text{ref}}[k]$ over the horizon with both a running and terminal cost:

$$\begin{aligned} \text{Cost} = & 100 \left(y[N_h] - y^{\text{ref}}[N_h]\right)^\top \left(y[N_h] - y^{\text{ref}}[N_h]\right) + \\ & + \sum_{i=0}^{N_h-1} 0.1 \left(y[i] - y^{\text{ref}}[i]\right)^\top \left(y[i] - y^{\text{ref}}[i]\right) \end{aligned} \tag{4.46}$$

In the LMPC case, $y[i] = C_L x_L[i]$ where x_L is the four dimensional system state and C_L is the projection matrix that isolates the states describing the current laser dot coordinates. In the K-MPC case, $y[i] = Cz[i]$, where C is defined as in (5.30).

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 (4.38) 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 (4.39) at a rate of 10 Hz. In all four controllers, the inputs are constrained to be in $[0, 10]$, since a voltage outside of this interval is not a valid command signal into the pressure regulators.

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

1. Task 1: Pac-Man (90 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:

$$\text{Error} = \frac{\sum_{i=1}^{N_{\text{steps}}} \sqrt{\left(y[i] - y^{\text{ref}}[i]\right)^\top \left(y[i] - y^{\text{ref}}[i]\right)}}{N_{\text{steps}}} \tag{4.47}$$

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. 4.4, 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, optimization problems were solved online repeatedly in less than 0.083 seconds for a 2 second receding horizon on a computer with 16 GB RAM and a 3.6 GHz CPU, requiring no offline computations.

4.4.6 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 LMPC 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 results in some overshooting and oscillations about the desired trajectory. In some applications, the reduced accuracy of the K-NMPC controller may be preferable to the higher frequency behavior of the other controllers. It should be noted, however, that this higher frequency behavior could likely be reduced by tuning the MPC cost function parameters to eliminate overshoot. Therefore, the oscillatory behavior observed in these experiments does not necessarily reflect a fundamental feature of the proposed MPC algorithms themselves, but is likely just an artifact of this particular choice of cost function.

Considering that the standard deviation under repeated inputs as described in Section 4.4.2 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 LMPC error is 6.9×10^{-2} %, a full order of magnitude larger than that of the other controllers.

The poor performance of LMPC 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 should make it much more reliable in the presence of external disturbances.

4.5 Incorporating Loading Conditions into Koopman Models

When interacting with objects or the environment, biological continuum appendages often encounter diverse external forces during locomotion, as well as varied loading conditions during prehensile actions. These factors can significantly alter the dynamics of the system, thereby posing challenges to the accuracy and robustness of both modeling and control processes. Consequently, understanding their impact on the system's dynamics is critical for accurate modeling and effective control.

In this section, we extend the Koopman-based modeling and control methodology, as discussed in previous sections, by incorporating external loading conditions into our models.

By incorporating loads into the model as states, our approach is able to estimate loading online via an observer within the control loop (see Fig. 4.5). 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. Using this approach, we demonstrate real-time, fully autonomous control of a pneumatically actuated soft continuum manipulator. 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.

4.5.1 Load Estimation for Linear 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 them within the states of our Koopman-based lifted system model, and then constructing an online observer to estimate their values. This strategy utilizes the underlying model to infer the most likely value of the loading conditions given past

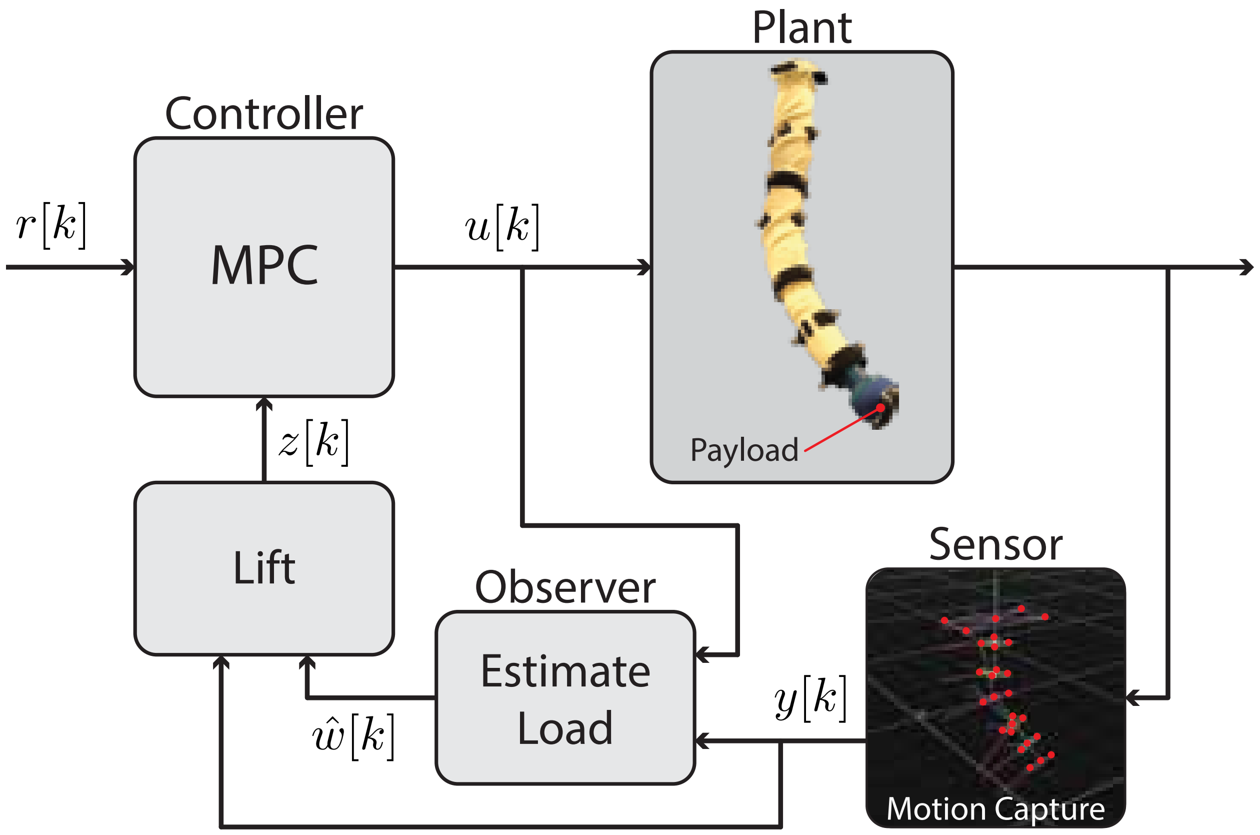

Figure 4.5: A soft continuum manipulator is tasked with following a reference trajectory $r[k]$ while carrying an unknown payload. At each time step, a model predictive controller computes the optimal input $u[k]$ to follow the reference trajectory based on a linear Koopman operator model and the position of the end effector $y[k]$, which is measured by a motion capture system. An observer computes a payload estimate $\hat{w}$ based on the previously measured inputs and outputs, which is then incorporated into the state of the model $z[k]$ via a lifting function.

input/output measurements.

Let $w \in \mathbb{R}^p$ be a parametrization of loading conditions, as defined by the user. For example, w might specify the mass at the end effector of a manipulator arm, the magnitude and direction of external forces, or the direction of gravity. We incorporate w into the state z using a new lifting function $\gamma : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^{(N-m)(p+1)}$, which accepts w as a second input and is defined as:

$$z = \gamma(y, w) = \begin{bmatrix} g(y) \\ g(y)w_1 \\ \vdots \\ g(y)w_p \end{bmatrix} = \underbrace{\begin{bmatrix} g(y) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & g(y) \end{bmatrix}}_{\Gamma(y)} \begin{bmatrix} 1 \\ w \end{bmatrix} \tag{4.48}$$

where $\Gamma(y) \in \mathbb{R}^{(N-m)(p+1)} \times \mathbb{R}^{p+1}$ is the matrix formed by diagonally concatenating $g(y)$, $p+1$ times, and w_i denotes the i^{th} element of w. Note that because this lifting function requires the loading condition w as an input, it must also be included in the snapshots $(a[k], b[k], u[k], w[k])$ to construct a Koopman model that accounts for loading.

In the data used for model construction we assume that w is known. However, in control applications w will often be unknown or impractical to measure directly. In that case, its value can be inferred based on the identified system model and past input/output measurements. We construct an observer that estimates the value of w at the j^{th} timestep by solving a linear least-squares problem using data from the N_w previous timesteps. Notice that the output at the j^{th} timestep $y[j]$ can be expressed in terms of the input $u[j-1]$, the output $y[j-1]$, and the load $w[j-1]$ at the previous timestep by combining the system model equations of (4.21) and then substituting (4.48) for $z[j-1]$,

$$y[j] = CAz[j-1] + CBu[j-1] \tag{4.49}$$

$$= CA\Gamma(y[j-1]) \begin{bmatrix} 1 \\ w[j-1] \end{bmatrix} + CBu[j-1]. \tag{4.50}$$

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

$$\begin{bmatrix} 1 \\ \hat{w}[j-1] \end{bmatrix} = \left(CA\Gamma(y[j-1])\right)^{\dagger} \left(y[j] - CBu[j-1]\right). \tag{4.51}$$

Under the assumption that the loading is equal to some constant $\tilde{w}$ over the previous N_w time steps, i.e. $w[i] = \tilde{w}$ 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,

$$\Lambda_A = \begin{bmatrix} CA\Gamma(y[j-1]) \\ \vdots \\ CA\Gamma(y[j-N_w]) \end{bmatrix} \tag{4.52}$$

$$\Lambda_B = \begin{bmatrix} y[j] - CBu[j-1] \\ \vdots \\ y[j-N_w+1] - CBu[k-N_w] \end{bmatrix} \tag{4.53}$$

Then, following from (4.51), the best estimate for $\bar{w}$ over the past N_w timesteps in the L^2-norm sense, denoted $\hat{w}$, is given by

$$\begin{bmatrix} 1 \\ \hat{w} \end{bmatrix} = \Lambda_A^{\dagger} \Lambda_B. \tag{4.54}$$

4.5.2 Control with Load Estimation

Model predictive control (MPC) solves an optimal control problem over a finite receding horizon, subject to system model constraints, to determine the next control action [113]. MPC can be applied to synthesize control inputs for linear time-varying systems using a convex quadratic programming (QP), wherein ones solves for the optimal sequence of control inputs over a finite

Algorithm 5: Koopman MPC with Load Estimation

```
Input:  Prediction horizon: $N_h$
        Model matrices: $A, B, C$
for  $k = 0, 1, 2, ...$ do
    if $k \mod N_e = 0$ then
        Estimate $\hat{w}'[j]$ via (4.54)
        $\hat{w}[k]$ = mean($\hat{w}'[j], \hat{w}[k-1], \ldots, \hat{w}[k-N_r]$)
        $j = j + 1$
    else
        $\hat{w}[k] = \hat{w}[k-1]$
    end
    Step 1: Solve QP to find optimal input $(u[i]^*)_{i=0}^{N_h}$
    Step 2: Set $u[k] = u[0]^*$
    Step 3: Apply $u[k]$ to the system
end
```

prediction horizon. The remainder of this section describes the formulation of the integration of MPC and load estimation with the Koopman-based models.

For the Koopman-based system models that incorporate loading, the knowledge of loading conditions w is required because MPC involves predicting system behavior. For the tasks in which the loading conditions are unknown, an estimate of the loading conditions $\hat{w}$ can be obtained periodically using the method described in Section 4.5.1. We integrate the MPC and the observer into a single control loop. Algorithm 5 summarizes the closed-loop operation of this Koopman-based MPC controller with these periodic load estimation updates. In particular, for systems with relatively stable loading conditions, it is not computationally efficient to compute a new 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.

4.6 Application: Pneumatic Soft Robotic Manipulator with Payload

To demonstrate and evaluate the proposed modeling and control approach in the previous section, we applied it to a real soft continuum robot arm that is capable of picking-up objects and moving its

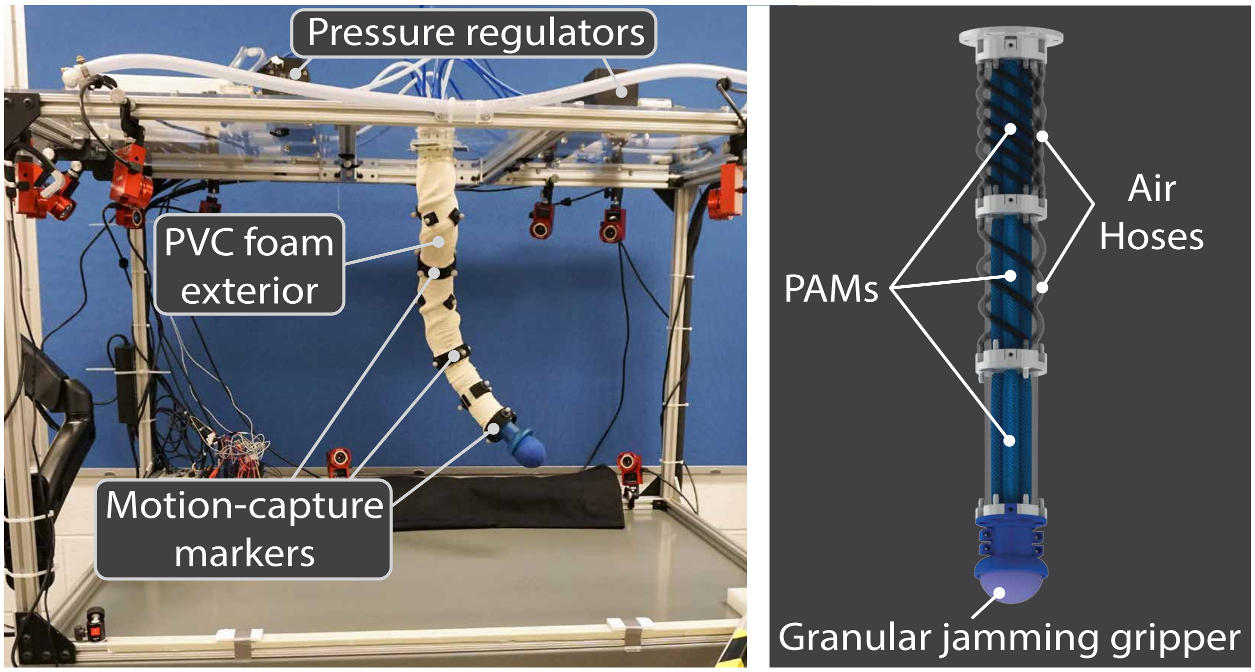

Figure 4.6: The soft robot arm consists of three bending sections, each actuated by three pneumatic artificial muscles (PAMs). The actuators are surrounded by a sleeve of flexible PVC foam, and pressurized air is supplied to the actuators via air hoses that wind around the exterior. The end effector consists of a granular jamming vacuum gripper [7], which is connected to a vacuum pump by a hose that runs along the interior of the arm.

end effector in 3D space. In the rest of this section, we describe in detail the robotic hardware, data collection and processing involved in the system identification process as well as the experiment setup by which performance was evaluated. Footage from these experiments is included in a supplementary video file.

4.6.1 System Identification of Soft Robot Arm

The soft robot arm used in experiments, shown in Fig. 4.6, is made up of three pneumatically actuated bending sections and an end effector comprised of a granular jamming vacuum gripper [7]. The interior of each section is composed of three pneumatic artificial muscles (PAMs) [132], which are attached to a central spine consisting of an air hose encased in flexible PVC foam tubing. The exterior of each section is composed of a much larger sleeve of flexible PVC foam surrounding the actuators, which serves to dampen high frequency oscillations and make the body of the arm softer overall. The pressures inside the actuators are 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 the actuators by air hoses that wrap around the outside of the foam sleeves. Each pressure regulator regulates one actuator.

To evaluate the performance of the proposed approach in consideration of system's stochasticity, we quantified the stochastic behavior of our soft robot system by observing the variations in output from period-to-period under sinusoidal inputs. The period of the sinusoidal inputs is 10 seconds, and the sampling time T_s is 0.083 seconds. Zero-order-hold is applied 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. Taking it into account, we expect only to be able to control the end effector position within ≈ 1 cm of a desired trajectory.

To construct a dynamic model, we chose the command voltages into the 9 pressure regulators, restricted to $[0, 10]^9$, as the input. We chose the Cartesian coordinates of the end of each bending section that lives in $\mathbb{R}^9$, with the last three coordinates corresponding to the end effector position as the system output, which were measured by the retro-reflective markers attached to the exterior of the robot tracked using a commercial OptiTrack motion capture system. And we chose the mass of the object, living in $\mathbb{R}_+$, held by the gripper as the parametrization of the loading condition.

Data for construction models was over 49 trials each lasting approximately 10 minutes, where 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 [136], a linear Koopman model that *does not* take loading into account using the approach described in Section 4.2.3, and a linear Koopman model that *does* incorporate loading using the approach from Section 4.5.1. Each of theses models was fit using the same set of $325, 733$ randomly generated data points just once, independent of any specific task.

Particularly, the linear state-space model provides a baseline for comparison and was identified using the MATLAB System Identification Toolbox [83]. And the 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 $V = 325,732$ snapshot pairs $\{a[v], b[v], u[v]\}_{v=1}^{V}$ that incorporate a single delay:

$$a[v] = \begin{bmatrix} y[v]^\top & y[v-1]^\top & u[v-1]^\top \end{bmatrix}^\top \tag{4.55}$$

$$b[v] = \begin{bmatrix} \phi_{T_s}(y[v])^\top & y[v]^\top & u[v]^\top \end{bmatrix}^\top \tag{4.56}$$

Since one delay is included, the dimension of each snapshot is $2n + m = 2(9) + 9 = 27$. We employed a lifting function $\psi : \mathbb{R}^{27} \times \mathbb{R}^{9} \to \mathbb{R}^{111}$ that was defined as,

$$\psi(y^d[v], u[v]) = \begin{bmatrix} g(y^d[v]) \\ u[v] \end{bmatrix} \tag{4.57}$$

where denote by $y^d[v]$ one of $a[v]$ and $b[v]$, i.e. $y^d[v] \in \{a[v], b[v]\}$. Note that the lifting function ψ has dimension $N = 111$, wherein the range of g has dimension 102, $g_i(y^d[v]) = y_i^d[v]$ for $i = 1, ..., 27$, and the remaining 75 basis functions $\{g_i : \mathbb{R}^{27} \to \mathbb{R}\}_{i=28}^{102}$ are monomials of maximum degree 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 in snapshot $\{a[v], b[v], u[v], w[v]\}_{v=1}^{V}$. The lifting function $\psi : \mathbb{R}^{27} \times \mathbb{R}^{9} \to \mathbb{R}^{231}$ was defined as,

$$\psi(y^d[v], u[v], w[v]) = \begin{bmatrix} \gamma(y^d[v], w[v]) \\ u[v] \end{bmatrix} = \begin{bmatrix} g(y^d[v]) \\ g(y^d[v])w[v] \\ u[v] \end{bmatrix} \tag{4.58}$$

where the range of g has dimension 111, $g_i(y^d[k]) = y_i^d[k]$ for $i = 1, ..., 27$, and the remaining 84

Table 4.3: 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**

basis functions $\{g_i : \mathbb{R}^n \to \mathbb{R}\}_{i=28}^{111}$ are monomials of maximum degree 2.

4.6.2 Description of Controllers

Corresponding to the three models, three model predictive controllers were constructed using control method described in the previous section. Each controller uses one of the identified models to compute online predictions and is referred to an acronym as follows,

- 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 [54] and run in closed-loop at 12 Hz with an MPC horizon $N_h = 12$.

4.6.3 Experiment 1: Trajectory Following with Known Payload

We first evaluated the performance of the three controllers when the true values of payloads at the end effector are given. We ran six trials in which the soft robot arm moves its end effector to try to follow a three-dimensional reference trajectory with payloads of 25, 75, 125, 175, 225, and 275 grams. The comparison of the reference and the actual paths of the end effector, and the tracking error over time for 3 of the trials are displayed in Fig. 4.7, and the RMSE tracking error for all 6 trials is shown in Table 4.3.

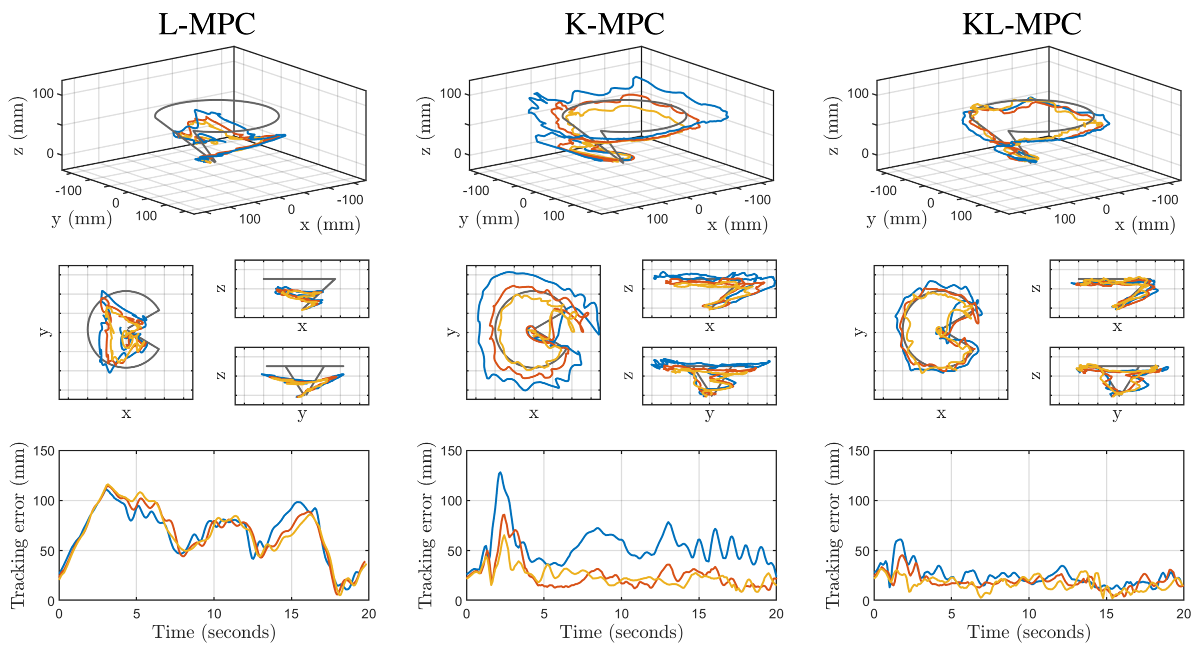

Figure 4.7: Experiment 1 Results: The end effector trajectories for the L-MPC (left), K-MPC (center), and KL-MPC (right) controllers when the true value of the payload is known. Trajectories corresponding to a payload of 25g are shown in blue, trajectories with a payload of 125g are shown in red, trajectories with a payload of 225g are shown in yellow, and the reference trajectory is shown in grey.

4.6.4 Experiment 2: Trajectory Following with Unknown Payload

To evaluate the efficacy of the combined control and load estimation method summarized by Algorithm 5, the manipulator was tasked with tracking a periodic reference trajectory that is not in the training data with the KL-MPC controller when the payload is unknown. New estimates were calculated every $N_e = 12$ time steps by solving (4.54) using measurements from the previous $N_w = 30$ timesteps, and $\hat{w}$ was computed by averaging over the most recent $N_r = 360$ estimates. Three trials were conducted with payloads of 25, 125, and 225 grams. The periodic reference trajectory was a circle with diameter of 200mm. Results of this experiment are shown in Fig. 4.8.

4.6.5 Experiment 3: Automated Object Sorting (Pick and Place)

With the same combined control and load estimation method used in experiment 2, an automated object sorting was performed on the soft manipulator. We placed five cups in front of the manipu-

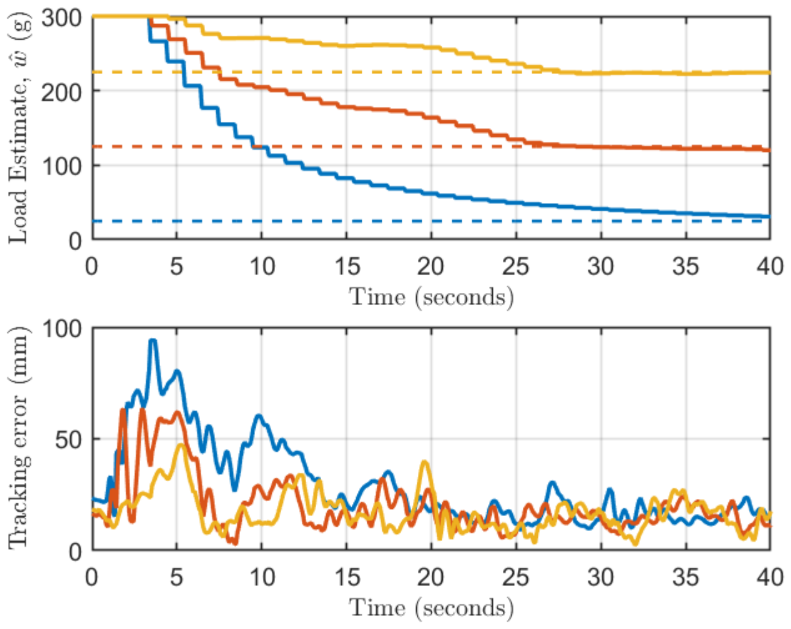

Figure 4.8: Experiment 2 Results: Periodic trajectory following with an unknown payload. The payload estimate over time (top) and tracking error over time (bottom) are shown for three trials with payloads of 25g, 125g, and 225g. Results for the 25g payload trial are shown in blue, results for the 125g payload trial are shown in red, and results for the 225g payload trial are shown in yellow.

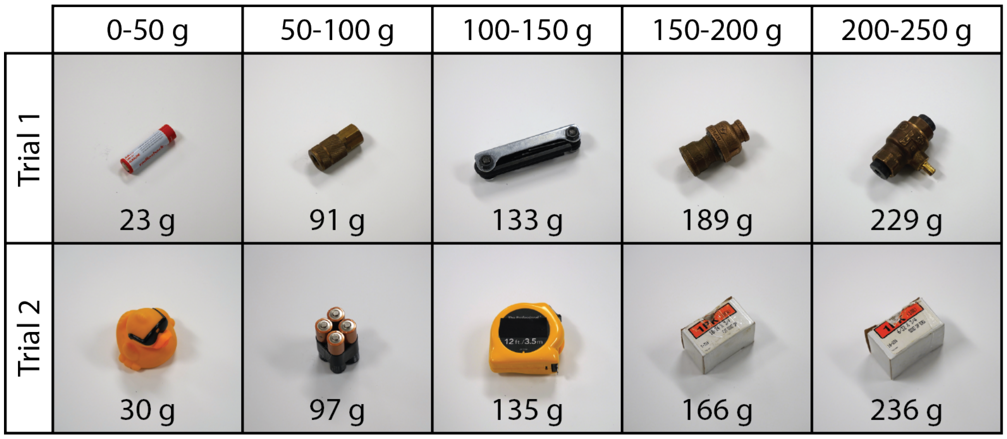

Figure 4.9: Objects used for Experiment 3: In each trial, the soft manipulator sorted a set of five objects according to their mass, based on an estimate computed by the online observer described in Section 4.5.1. The set of objects used for each trial are separated by row, and the mass of each object is written below it.

lator, each corresponding to a 50 gram interval between 0 and 250 grams (i.e. 0-50, 50-100, etc.). We selected objects with mass between 0 and 250 grams. Given one of these objects, the task was to place the object into the cup that corresponds to its mass. Note that objects with mass above 250 grams weren't used because such loads reduce the robot workspace too much.

For each trial, a human feeds an object of unknown mass into the gripper, then the manipulator runs KL-MPC to estimate the payload (Algorithm 5) 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. 4.9). The objects used and their masses are shown in Fig. 4.9.

4.6.6 Discussion

This section presents an extension of the Koopman-operator based approach to enable consistent control performance of a soft continuum manipulator arm under variable loading conditions.

The experimental findings underscore the effectiveness of integrating load information directly into the modeling and control algorithm, leading to substantial improvements in tracking accuracy and robustness under diverse loading scenarios.

In Experiment 1, the KL-MPC controller, which incorporated the payload value, reduced the RMSE tracking error averaged over all payloads by approximately 33% and the standard deviation by about 77% compared to the K-MPC controller, which did not utilize information about the payload. And the KL-MPC controller reduced the RMSE tracking error averaged over all payloads by approximately 71% compared to the L-MPC controller. (see Table 4.3). This illustrates that the KL-MPC controller is more accurate and precise than the other two controllers.

By combining the estimation, modeling, and control into a single MPC algorithm (Algorithm 5), experiment 2 showed that the load estimate gradually become more accurate and the tracking error was reduced as the controller attempted to follow a circular trajectory, without needing to inform the KL-MPC controller of the actual load value. 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 pick and place object manipulation using the same algorithm. Unknown objects were successfully sorted by mass, which requires the payload estimate to be accurate enough to choose the correct container for each object and the tracking error of the "drop-off" trajectory to be small enough not to miss the cup.

These experiments collectively demonstrate that the Koopman-based modeling and control framework can generate dynamical models for continuum robotic systems that are robust against changes in external loading conditions. This robustness is crucial for studying biological continuum appendages, which often experience a range of external forces during movement and grasping activities. Thus, the framework offers a way to accurately represent the complex dynamics of biological continuum appendages while significantly reducing the computational effort required to analyze and control these appendages, thereby enabling a comprehensive exploration of the biological appendage's functionality in a wide array of scenarios, ranging from interaction-free environments to those involving complex external loading conditions and beyond.

CHAPTER 5

Bilinear Koopman Realizations for Data-driven Modeling and Control

5.1 Introduction

As elaborated in the previous chapter, a global linearization technique based on Koopman operator theory can convert a nonlinear dynamical system into an equivalent linear system. This is achieved by lifting the system into a higher-dimensional space of scalar-valued functions called *observables*. In this space, the evolution of observables along trajectories of the nonlinear system are described by the (linear) Koopman operator [27]. The Koopman operator captures the behavior of the system everywhere, not just near equilibrium points, providing a global linear representation of the system in terms of observables.

However, despite their linearity in the space of observables, Koopman representations have limitations that hinder their ability to inform the control of arbitrary nonlinear dynamical systems. One limitation is that linearity with respect to observables does not imply linearity with respect to the control input. Therefore, Koopman models are not necessarily compatible with computationally efficient linear control design techniques such as LQR [9] and linear MPC [113]. A further limitation of Koopman linear embeddings is that they are in general infinite-dimensional. Therefore, finite-dimensional truncations must be used in practice, which merely approximate the behavior of the original nonlinear system rather than capture it perfectly.

Linearity with respect to the control input can be enforced by restricting the Koopman operator to a subspace of observables which only includes linear functions of the control input. This strategy is introduced in [69] to construct linear predictors of nonlinear systems for model predictive control. This and similar approaches have shown capable of achieving better control performance than local linearization techniques on a variety of different robots such as a Sphero SDK [3], a swimming fish robot [82], a Rethink Sawyer robot arm [4], and several soft robots [24, 23].

Such applications make use of finite-dimensional matrix approximations of the Koopman operator that are identified from data using a linear system identification technique known as Extended Dynamic Mode Decomposition (EDMD) [138, 86]. These approximations rely upon the implicit assumption that the identified Koopman matrix converges to the true Koopman operator as its dimension goes to infinity. In other words, they assume that sufficient accuracy can always be achieved by a Koopman model with linear control inputs if its dimension is large enough. However, as shown in [14], a valid Koopman representation is not guaranteed to exist when the approximation is restricted to a subspace that excludes nonlinear functions of the input. This holds true even if the subspace is infinite-dimensional.

There is thus a trade-off when looking to utilize Koopman representations for control of a system with unknown dynamics. Koopman representations with linear control inputs are advantageous because they are compatible with computationally efficient linear control techniques. However, a valid Koopman representation of this form may not exist, in which case finite-dimensional approximations of this type are unlikely to improve with dimension. On the other hand, every system admits a valid Koopman representation with nonlinear control inputs. However, such representations have limited usefulness for control applications because they are not compatible with efficient control techniques.

Koopman representations with bilinear control input terms strike a compromise between these two extremes. Such models preserve some of the computational benefits of linear Koopman models, while being more likely to exist for arbitrary dynamical systems. This makes them desirable for control applications.

Previous work has investigated Koopman-based bilinearization and control methods for nonlinear systems. In [52], the Koopman Canonical Transform (KCT) is used to describe how to construct bilinear realizations over the set of Koopman eigenfunctions. It also introduces a set of sufficient conditions for a nonlinear system to be bilinearizable. In [103], it is shown that the control-affine property of dynamical systems is preserved for the continuous Koopman operator, which allows bilinear surrogate models to be constructed by interpolating between different operators. These bilinear models are incorporated into a model predictive control framework and applied to the control of several nonlinear systems.

In line with previous work, this chapter explores the benefits of utilizing the Koopman operator to construct bilinear models for systems with unknown dynamics. While it is feasible to develop approximate Koopman model realizations with linear, bilinear, or nonlinear control inputs directly from data, we suggest that bilinear Koopman realizations, when identified through this approach, are likely to provide better predictions than their linear counterparts and offer greater computational efficiency compared to the nonlinear options

In this chapter, we first offer a theoretical justification for the proposed advantages of bilinear Koopman model realizations. We specify necessary and sufficient conditions for a dynamical system to have a valid linear or bilinear realization over a given set of observables, and use these conditions to show that every control-affine system admits a bilinear realization, but does not necessarily admit a linear one. Then, we provide instructions on how to construct approximate bilinear Koopman model realizations from data. Using this approach, we construct several Koopman model realizations of a simulated robot arm, and show that a bilinear realization simultaneously exceeds the prediction accuracy of a linear realization and the computational efficiency of a nonlinear realization when incorporated into a model predictive control framework.

5.2 Methods

This section introduces a definition of bilinear realizations based on the Koopman operator and details the necessary and sufficient conditions for a realization to be considered valid. To enable a direct comparison between the linear realizations presented in the previous chapter and the bilinear model realization, key definitions of the linear realization are briefly revisited, alongside a similar analysis of the necessary and sufficient conditions for a linear realization to be feasible. We reveal that while every control-affine system can be represented by a bilinear realization, which may be infinite-dimensional, a linear realization is not always feasible

5.2.1 Necessary and Sufficient Conditions for Bilinear Linearizations

Consider the same nonlinear dynamical system form as presented in the previous chapter

$$\dot{x}(t) = F(x(t), u(t)) \tag{5.1}$$

where $x(t) = [x_1(t), \ldots, x_n(t)]^\top \in X \subset \mathbb{R}^n$ is the state and $u(t) = [u_1(t), \ldots, u_m(t)]^\top \in U \subset \mathbb{R}^m$ is the input of the system at time $t \in [0, +\infty)$, F is a continuously differentiable function, and X, U are compact subsets.

Let $\mathcal{F}$ be the infinite-dimensional function space composed of all square-integrable real-valued functions with compact domain $X \times U \subset \mathbb{R}^{n \times m}$. Elements of $\mathcal{F}$ are called *observables*. Then 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 every observable $f \in \mathcal{F}$ along the trajectories of the system.

As detailed in the previous chapter, by choosing a countable set of observables $\{f_i \in \mathcal{F}\}_{i=1}^N$ (where $N \in \mathbb{N} \cup \infty$) from which the original state can be recovered through an inverse mapping $C : \mathbb{R}^N \to \mathbb{R}^n$, one can generate model realizations of the dynamical system (5.1), which generates the same state response x as (5.1) under any input signal u. Taken together, this set of observables constitutes the *lifted state* of the realization. The evolution of each component of the lifted state is described by applying the continuous Koopman operator, and the output equation of the realization consists of the mapping C:

$$\frac{d}{dt} f_i(t) = \mathcal{K} f_i(t) \qquad \text{for } i = 1, \dots, N \tag{5.2}$$

$$x(t) = C\left(f_1(t), \dots, f_N(t)\right) \tag{5.3}$$

where $f_i(t)$ is shorthand for $f_i(x(t), u(t))$. We say that the realization is defined over the set of observables $\{f_i\}_{i=1}^N$. Note that model realizations are not unique since they depend on the choice of observables. Certain types of realizations are particularly amenable to control design since the control input appears either linearly or bilinearly in them. Before formally defining these realization, we first define the following subsets of the space of all observables $\mathcal{F}$ for notational convenience:

$$\mathcal{X} := \{f \in \mathcal{F} \,|\, f(\tilde{x}, \tilde{u}) = \tilde{x}_i \text{ for some } i = 1, ..., n\} \tag{5.4}$$

$$\mathcal{U} := \{f \in \mathcal{F} \,|\, f(\tilde{x}, \tilde{u}) = \tilde{u}_i \text{ for some } i = 1, ..., m\} \tag{5.5}$$

$$\mathcal{Z} := \{f \in \mathcal{F} \,|\, f(\tilde{x}, \tilde{u}_1) = f(\tilde{x}, \tilde{u}_2)\,, \forall \tilde{u}_1, \tilde{u}_2 \in \mathbb{R}^m\} \tag{5.6}$$

where $\tilde{x} = [\tilde{x}_1, \dots, \tilde{x}_n] \in X$ and $\tilde{u} = [\tilde{u}_1, \dots, \tilde{u}_m] \in U$. In other words, $\mathcal{X}$ is the set of functions that project onto components of the state, $\mathcal{U}$ is the set of functions that project onto components of the input, and $\mathcal{Z}$ is the set of all functions that depend on the state only, including constant functions. With this notation in hand, we define linear and bilinear model realizations as follows:

Definition 1 (Linear and bilinear model realizations)**.** *A model realization of* $\dot{x}(t) = F(x(t), u(t))$ *over the set of observables* $\{z_i \in \mathcal{Z}\}_{i=1}^N$ *is **bilinear** if there exist sets of coefficients* $\{a_{ij} \in \mathbb{R}\}_{i=1,j=1}^{N,N}$, $\{b_{ij} \in \mathbb{R}\}_{i=1,j=1}^{N,m}$, $\{h_{ijk} \in \mathbb{R}\}_{i=1,j=1,k=1}^{N,N,m}$, *and* $\{c_{ij} \in \mathbb{R}\}_{i=1,j=1}^{n,N}$ *such that*

$$\frac{d}{dt} z_i(t) = \sum_{j=1}^{N} a_{ij} z_j(t) + \sum_{j=1}^{m} b_{ij} u_j(t) + \sum_{j=1}^{m} \sum_{k=1}^{N} h_{ijk} z_k(t) u_j(t) \tag{5.7}$$

for $i = 1, \dots, N$ and

$$x_i(t) = \sum_{j=1}^{N} c_{ij} z_j(t) \tag{5.8}$$

*for $i = 1, \dots, n$ where $z_i(t)$ is shorthand for $z_i(x(t), u(t))$, $x(t) = [x_1(t), \dots, x_n(t)]^\top$, and $u(t) = [u_1(t), \dots, u_m(t)]^\top$. If $h_{ijk} = 0$ for all i, j, k, then the realization is said to be **linear**.*

The lifted state of these realizations consists entirely of observables that depend on the state only, and the input appears only in linear and bilinear terms.

As stated previously, realizations are not unique. They depend on the choice of observables over which they are defined. For a particular choice of observables to yield a bilinear or linear realization they must satisfy certain conditions. These conditions are presented in the following theorem.

Theorem 1 (Necessary and sufficient conditions for linear and bilinear realizations)**.** *The realization of the system governed by* (5.1) *over a set of observables $\bar{\mathcal{Z}} = \{z_i \in \mathcal{Z}\}_{i=1}^N$ is:*

1. ***Bilinear** if and only if*

$$\frac{\partial z_i}{\partial x} F \in \mathit{span}\left(\bar{\mathcal{Z}} \cup \mathcal{U} \cup \{f \cdot g | f \in \bar{\mathcal{Z}}, g \in \mathcal{U}\}\right) \tag{5.9}$$

   for $i = 1, \dots, N$ and $\mathcal{X} \subset \mathit{span}(\bar{\mathcal{Z}})$.

2. ***Linear** if an only if*

$$\frac{\partial z_i}{\partial x} F \in \mathit{span}\left(\bar{\mathcal{Z}} \cup \mathcal{U}\right) \tag{5.10}$$

   for $i = 1, \dots, N$ and $\mathcal{X} \subset \mathit{span}(\bar{\mathcal{Z}})$.

Note that linear systems are by definition also bilinear.

Proof. The definition for both realizations specifies that the state can be recovered as a linear combination of the observables, i.e.

$$x_i = \sum_{j=1}^{N} c_{ij} z_j, \qquad \text{for } i = 1, \ldots, n \tag{5.11}$$

$$\Leftrightarrow x_i \in \text{span}\left(\bar{\mathcal{Z}}\right), \qquad \text{for } i = 1, \ldots, n \tag{5.12}$$

$$\Leftrightarrow \mathcal{X} \subset \text{span}(\bar{\mathcal{Z}}). \tag{5.13}$$

The conditions specified by (5.9) and (5.10) are verified by applying the chain rule to the left hand side of (5.7):

$$\frac{d}{dt} z_i = \frac{\partial z_i}{\partial x} \frac{dx}{dt} = \frac{\partial z_i}{\partial x} F, \qquad \text{for } i = 1, \ldots, N \tag{5.14}$$

Plugging (5.14) into the definition of a bilinear realization from Def. 1 then yields,

$$\frac{\partial z_i}{\partial x} F = \sum_{j=1}^{N} a_{ij} z_j + \sum_{j=1}^{m} b_{ij} u_j + \sum_{j=1}^{m} \sum_{k=1}^{N} h_{ijk} z_k u_j \tag{5.15}$$

$$\Leftrightarrow \frac{\partial z_i}{\partial x} F \in \text{span}\left(\bar{\mathcal{Z}} \cup \mathcal{U} \cup \{f \cdot g | f \in \bar{\mathcal{Z}}, g \in \mathcal{U}\}\right) \tag{5.16}$$

Similarly, plugging (5.14) into the definition of a linear realization from Def. 1 yields,

$$\frac{\partial z_i}{\partial x} F = \sum_{j=1}^{N} a_{ij} z_j + \sum_{j=1}^{m} b_{ij} u_j \tag{5.17}$$

$$\Leftrightarrow \frac{\partial z_i}{\partial x} F \in \text{span}(\bar{\mathcal{Z}} \cup \mathcal{U}) \tag{5.18}$$

□

Since there are computational benefits to performing control synthesis with linear and bilinear realizations, we would prefer to choose sets of observables that admit a realization of one of these types. However, unless the dynamics of the system are known, it is not possible to verify whether a particular set of observables satisfies the conditions outlined in Theorem 1. Nevertheless, as we show below, one can prove that for control-affine systems, a realization over $\mathcal{Z}$ is bilinear.

Corollary 1. *If the system governed by* (5.1) *is control-affine and a set of observables* $\bar{\mathcal{Z}} = \{z_i \in \mathcal{Z}\}_{i=1}^{\infty}$ *is a basis of* $\mathcal{Z}$*, then the realization of the system defined over* $\bar{\mathcal{Z}}$ *is bilinear.*

Proof. The system being control-affine implies that there exists $F_x : X \to \mathbb{R}^n$ and $\{F_u : X \to \mathbb{R}^n\}_{i=1}^m$ such that

$$F(x,u) = F_x(x) + \sum_{j=1}^{m} F_u^j(x) u_j \tag{5.19}$$

Consider a realization generated by the Koopman operateor as in (5.3), then the function $\frac{\partial z_i}{\partial x}$ maps from X to $\mathbb{R}^{1\times n}$. Thus, the products of $\frac{\partial z_i}{\partial x}$ with F_x and F_u^j map from X to $\mathbb{R}$, i.e. they are in $\mathcal{Z}$. Given that $\bar{\mathcal{Z}}$ is a basis for $\mathcal{Z}$, it follows that

$$\frac{\partial z_i}{\partial x} F_x \in \text{span}\{\bar{\mathcal{Z}}\}, \qquad \forall i \tag{5.20}$$
$$\frac{\partial z_i}{\partial x} F_u^j \in \text{span}\{\bar{\mathcal{Z}}\}, \qquad \forall i, j \tag{5.21}$$

This implies that for some set of coefficients $\{\gamma_{ij}\}_{i=1,j=1}^{\infty,m}$,

$$\frac{\partial z_i}{\partial x} F_u^j = \sum_{i=1}^{\infty} \gamma_{ij} z_i, \qquad \forall j \tag{5.22}$$
$$\implies \frac{\partial z_i}{\partial x} F_u^j u_j = \sum_{i=1}^{\infty} \gamma_{ij} z_i u_j, \qquad \forall j \tag{5.23}$$
$$\implies \frac{\partial z_i}{\partial x} F_u^j u_j \in \text{span}\left(\{f \cdot g | f \in \bar{\mathcal{Z}}, g \in \mathcal{U}\}\right), \qquad \forall j \tag{5.24}$$

Thus, (5.9) is satisfied. Additionally, the condition that $\mathcal{X} \subset \text{span}(\bar{\mathcal{Z}})$ is satisfied since $\mathcal{X} \subset \mathcal{Z}$ and $\mathcal{Z} = \text{span}(\bar{\mathcal{Z}})$. □

A consequence of the preceding corollary is that every control-affine system has a valid bilinear realization over an infinite set of basis functions. Thus, generic sets of basis functions (i.e. polynomial, Fourier, or radial) can be used as the chosen set of observables in $\bar{\mathcal{Z}}$ since linear combinations of them can represent arbitrary functions in $\mathcal{Z}$. It should be noted that the theorem offers no such guarantee that a system has a valid linear realization, no matter the choice of basis functions. Therefore, realizations defined over a finite set of basis functions of the space of state-dependent observables will converge to a valid bilinear realization as the number of basis functions goes to infinity, but will not necessarily converge to a valid linear realization.

5.2.2 Bilinear Model Realization

With a suitable choice of observables $\{\psi_i\}_{i=1}^{N(m+1)+m}$, the Koopman matrix $\bar{\mathcal{K}}$ can be constructed such that it is decomposable into a bilinear system representation. The first N observables are defined as functions of the state only, the next Nm observables 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 projections onto each component of the input, i.e.

$$\psi_i := \begin{cases} z_i \in \mathcal{Z}, & \text{for } i = 1, \dots, N \\ z_{i-N} \cdot \pi_{u_1}, & \text{for } i = N+1, \dots, 2N \\ \quad \vdots & \qquad \vdots \\ z_{i-Nm} \cdot \pi_{u_m}, & \text{for } i = Nm+1, \dots, N(m+1) \\ \pi_{u_{i-N(m+1)}} \in \mathcal{U}, & \text{for } i = N(m+1)+1, \dots, \\ & \qquad N(m+1)+m \end{cases} \tag{5.25}$$

where π_{u_i} denotes the projection onto the i^{th} component of the input.

The Koopman matrix can be identified from data according the steps laid out in Section 4.2.2. Then, by construction, the coefficients for a bilinear realization of the form specified in (5.7) are embedded within the first N rows of the transpose of the Koopmn matrix in the following manner,

$$\bar{\mathcal{K}}^\top = \begin{bmatrix} A & H_1 & \cdots & H_m & B \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{5.26}$$

where A and B are defined as

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix}, \qquad B = \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{Nm} \end{bmatrix} \tag{5.27}$$

and H is defined as

$$H_i = \begin{bmatrix} h_{i11} & \cdots & h_{i1N} \\ \vdots & \ddots & \vdots \\ h_{iN1} & \cdots & h_{iNN} \end{bmatrix}. \tag{5.28}$$

For convenience, we can define the first n basis functions as projections onto each component of the state, i.e.

$$\psi_i := \pi_{x_i} \in \mathcal{X} \subset \mathcal{Z} \quad \text{for } i = 1, \ldots, n \tag{5.29}$$

Then, the coefficients of the output equations of (5.8) can be defined as simply

$$c_{ij} = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases} \tag{5.30}$$

5.2.3 Model Predictive Control

As introduced in the previous chapter, model predictive control algorithms optimally choose a sequence of control inputs given a desired output trajectory and system model [113]. 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.

Since the linear MPC optimization problem is convex, it has a unique globally optimal solution that can be efficiently computed without initialization even for high-dimensional models [21, 102, 129]. This is not the case for the nonlinear MPC problem which has nonlinear constraints that render it non-convex [6]. As a result, algorithms to solve such problems typically require initialization, can struggle to find globally optimal solutions [106], and take longer to solve per iteration. Though techniques have been proposed to improve the speed and accuracy of nonlinear MPC algorithms [101, 56, 149] they are still often unable to achieve the computational efficiency required for real-time control.

In this chapter we implement and compare three model predictive controller algorithms which we refer to by the following abbreviations:

- K-MPC: Koopman-based linear MPC
- K-BMPC: Koopman-based bilinear MPC
- K-NMPC: Koopman-based nonlinear MPC

The K-MPC and K-NMPC controllers are standard model predictive controllers that use linear and nonlinear Koopman model realizations to generate predictions, respectively, introduced in the previous chapter. The K-BMPC controller uses a bilinear Koopman model realization to generate predictions. However, bilinear constraints would render the problem non-convex, so we instead rely on a linear approximation constructed by fixing the value of the lifted state in the bilinear terms over the prediction horizon, N_h. The dynamics constraints in the K-BMPC optimization problem then become:

$$z_i[t+1] = \sum_{j=1}^{N} a_{ij} z_j[t] + \sum_{j=1}^{m} b_{ij} u_j[t] + \sum_{j=1}^{m}\sum_{k=1}^{N} h_{ijk} z_k[0] u_j[t] \tag{5.31}$$

$$= \sum_{j=1}^{N} a_{ij} z_j[t] + \sum_{j=1}^{m} \left(b_{ij} + \sum_{k=1}^{N} h_{ijk} z_k[0] \right) u_j[t] \tag{5.32}$$

for $i = 1, \dots, N$ and $t = 1, \dots, N_h$, where $[\cdot]$ denotes a discrete time index. The resulting linear dynamics approximate the behavior of the bilinear realization in a neighborhood of the initial lifted state $\{z_i[0]\}_{i=1}^{N}$. This introduces prediction error, but turns it into a convex quadratic program which can be solved quickly enough to be updated at every time-step.

Beyond the dynamics constraints, all three MPC controllers can accommodate additional linear state and input constraints. Constraints on nonlinear functions of the unlifted state $\vec{x}$ can be included as linear constraints as long as such functions are included in the set of observables over which a realization is defined.

The main difference between the K-BMPC controller and the other MPC 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 MPC problem.

It is important to note, however, that all finite-dimensional Koopman realizations constructed from data are mere approximations of the true dynamics of a system. Hence, even the optimal solutions to the linear/nonlinear MPC problems are not necessarily optimal with respect to the true dynamics of the system.

5.3 Experiments and Results

To compare the performance of linear, bilinear, and nonlinear Koopman realizations for modeling systems with unknown dynamics, we applied the methods from Section 5.2 to a collection of randomly generated dynamical systems as well as a simulated planar arm system.

The code used to generate these results can be found in a publicly accessible repository[1].

5.3.1 Description of systems

20 systems with one-dimensional input $u(t) \in \mathbb{R}$ and state $x(t) \in \mathbb{R}$ were generated with dynamics of the following form,

$$\dot{x}(t) = e^{-x(t)^4} \left(\sum_{i=1}^{3} \lambda_i x(t)^{\mu_i} u(t)^{\sigma_i} + \lambda_4 u(t) \right) - \tan^{-1}(x(t)) \tag{5.33}$$

where the coefficients $\lambda_i \sim \text{unif}(-1, 1)$ and the exponents $\mu_i, \sigma_i \sim \text{unif}\{0, 1, 2\}$ for each system were selected from the continuous and discrete uniform distributions, respectively. Note that these systems are not strictly control-affine, since higher order input terms are permitted. The exponential and inverse tangent terms ensure the state of each system remains finite under arbitrary inputs while still allowing for large variability in behavior.

The data used to identify Koopman realizations of each of the systems was a set of 10000 snapshots with a time-step of $T_s = 0.01$s collected over a range of randomized initial conditions and inputs.

In addition to these randomly generated systems, a simulated planar arm system was also considered (Fig. 5.3). 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,

$$\vec{u}(t) = [\tau_1(t),\, \tau_2(t),\, \tau_3(t)]^\top \tag{5.34}$$

$$\vec{x}(t) = [\alpha_1(t),\, \beta_1(t),\, \alpha_2(t),\, \beta_2(t),\, \alpha_3(t),\, \beta_3(t)]^\top \tag{5.35}$$

The data used to identify Koopman realizations of the arm system was a set of 12000 snapshots

[1]https://github.com/roahmlab/koopman-realizations.git

with a time-step of $T_s = 0.05$s collected over a range of randomized initial conditions and inputs.

5.3.2 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. We identified several models of each type on subspaces spanned by monomial basis functions.

The sets of basis functions used for identifying the linear models consisted of all monomials of the components of the state up to a specific degree denoted ρ, plus the projections onto each component of the input, i.e.

$$\begin{aligned} \{\psi_i(\tilde{x}, \tilde{u})\}_{i=1}^M = &\{\tilde{x}_1^{\rho_1} \cdots \tilde{x}_n^{\rho_n} | \rho_1 + \cdots + \rho_n \leq \rho\} \\ &\cup \{\tilde{u}_i | i \in \{1, ..., m\}\} \end{aligned} \tag{5.36}$$

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

$$\begin{aligned} \{\psi_i(\tilde{x}, \tilde{u})\}_{i=1}^M = \{(\tilde{x}_1^{\rho_1} \cdots \tilde{x}_n^{\rho_n})\hat{u} | \rho_1 + \cdots + \rho_n \leq \rho, \\ \hat{u} \in \{1, \tilde{u}_1, ..., \tilde{u}_m\}\} \end{aligned} \tag{5.37}$$

where $M = (N + 1)(m+!)$ and $N = (n + \rho)!/(n!\rho!)$. The sets of basis functions used for identifying nonlinear models consisted of monomials up to degree ρ of both the input and the state, i.e.

$$\begin{aligned} \{\psi_i(\tilde{x}, \tilde{u})\}_{i=1}^M = \{(\tilde{x}_1^{\rho_1} \cdots \tilde{x}_n^{\rho_n})(\tilde{u}_1^{\rho_{n+1}} \cdots \tilde{u}_m^{\rho_{n+m}})| \\ \rho_1 + \cdots + \rho_{n+m} \leq \rho\} \end{aligned} \tag{5.38}$$

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

For each of the 20 randomly generated systems, 20 linear models were identified for values of $\rho = \{1, .., 20\}$, 10 bilinear models were identified for values of $\rho = \{1, .., 10\}$, and 5 nonlinear

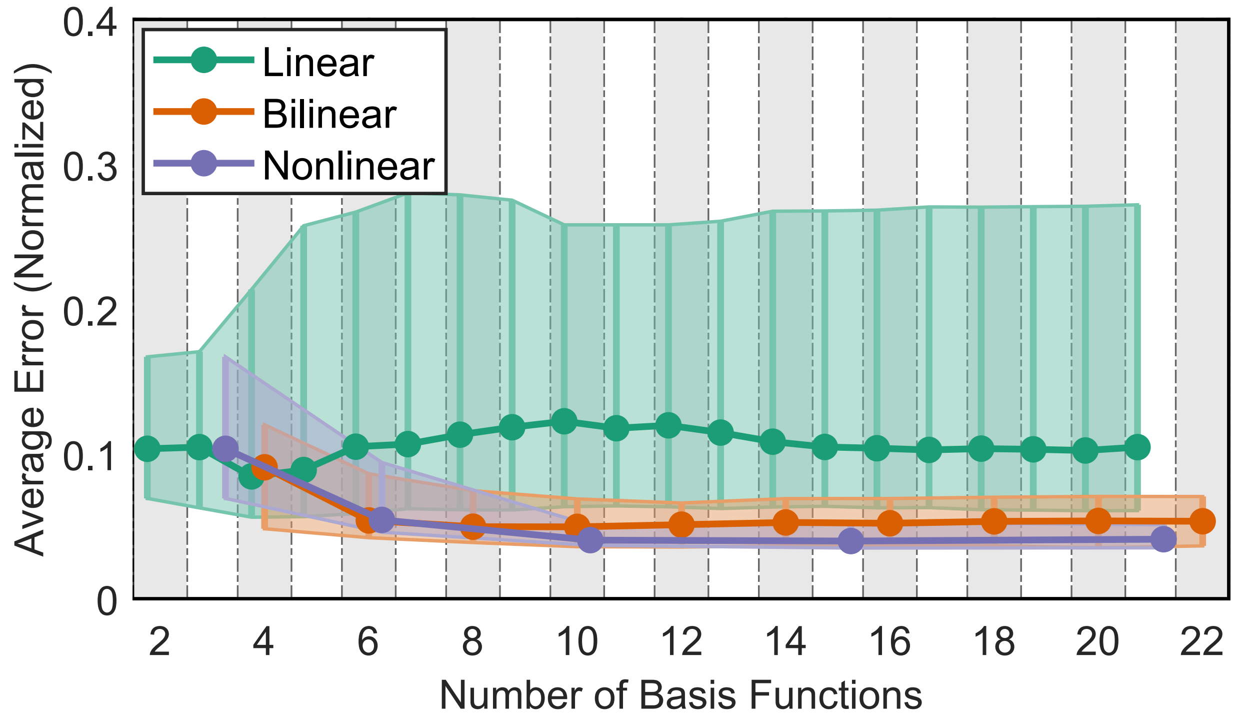

Figure 5.1: The model prediction error versus the number of basis functions for several linear, bilinear, and nonlinear realizations of 20 randomly generated systems. Dots indicate the median prediction error and the lower and upper bounds of the shaded regions signify the lower and upper quartiles, respectively.

Table 5.1: Number of Monomial Basis Functions for Models of Arm

ρ	# of basis functions		
	Linear	Bilinear	Nonlinear
1	10	28	10
2	31	112	55
3	87	336	220
4	213	840	715
5	465	-	-
6	927	-	-

models were identified for values of $\rho = \{1, .., 5\}$. The prediction accuracy of each model was evaluated by comparing a model simulation to validation data for a 10 second trial, computing the average error over all time-steps, then normalizing by the average error incurred by the zero response over all time-steps.

Fig. 5.1 displays the median prediction error over all random systems versus the number of basis functions used to identify the Koopman operator matrix, i.e. dim$(\psi(\tilde{x}, \tilde{u}))$. The median prediction error of the bilinear and nonlinear realizations generally decreases then plateaus as the number of basis functions increases, whereas it varies more erratically for the linear realizations.

For the planar arm system, 6 linear models were identified for values of $\rho = \{1, .., 6\}$, and 4 bilinear and nonlinear models were identified for values of $\rho = \{1, ..., 4\}$. Table 5.1 indicates the total number of basis functions, 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 for a 20 second trial and computing the average error over all time-steps. This error was quantified as the Euclidean distance between the predicted and real outputs in $\mathbb{R}^6$, normalized by the average Euclidean distance error incurred by the zero response over all time-steps.

Fig. 5.2 displays the prediction error versus the dimension of the Koopman operator matrix for each model. 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 and nonlinear models become more accurate as the dimension increases.

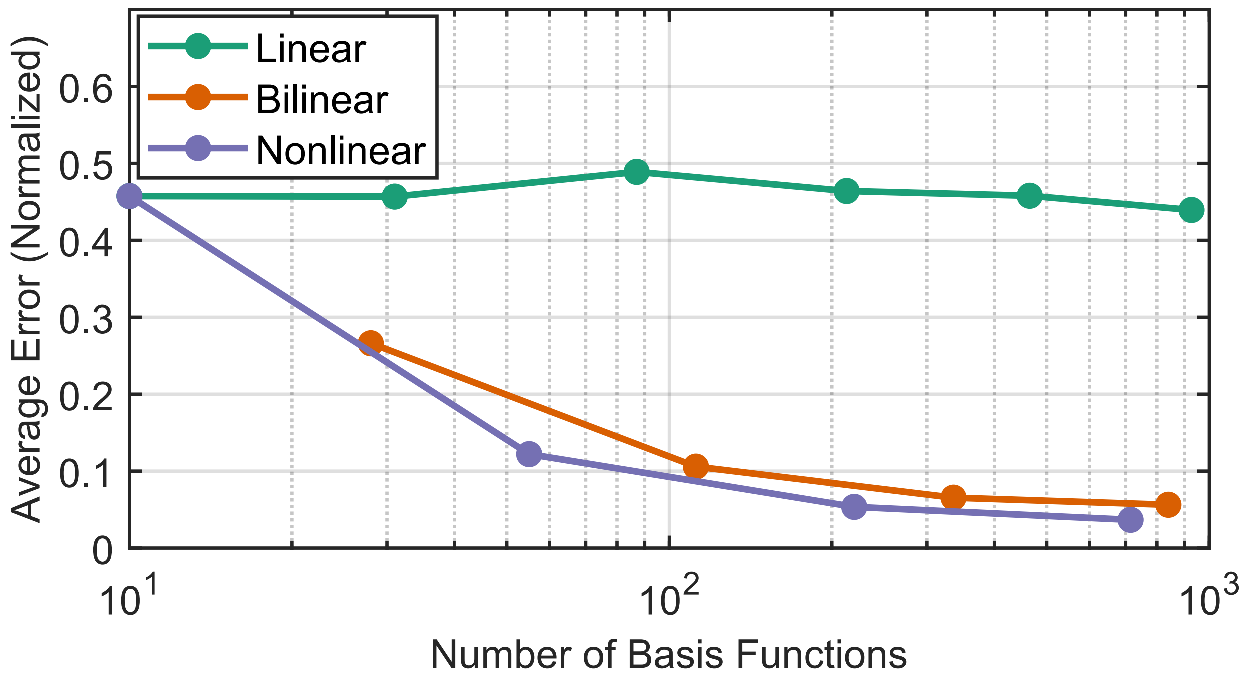

Figure 5.2: The model prediction error for several linear, bilinear, and nonlinear Koopman model realizations of the simulated arm system. As the number of basis functions increases, the error of the linear model changes little while the error of the bilinear and nonlinear models decrease monotonically.

5.3.3 Control Performance Comparison

To highlight the relative strengths and weaknesses of the Koopmans-based control techniques described in Section 5.3.2, we applied them to the simulated 3-link planar arm system. A K-MPC controller, K-BMPC controller, and K-NMPC controller was constructed from the linear, bilinear, and nonlinear model realizations identified in Section 5.3.2 for $\rho = 3$, respectively. Each controller was then employed to perform the same trajectory following task.

Each controller computed solutions over a $N_h = 10$ step horizon and had identical cost functions. 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 $(\alpha_3^{\text{ref}}, \beta_3^{\text{ref}})$ as well as the control effort at each time-step.

The control experiment was 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 traces out the shape of a block letter M over a time period of 15 seconds, starting from the robot's hanging position. Fig. 5.3 shows the path of the end effector using each of the controllers, and Fig. 5.4 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.

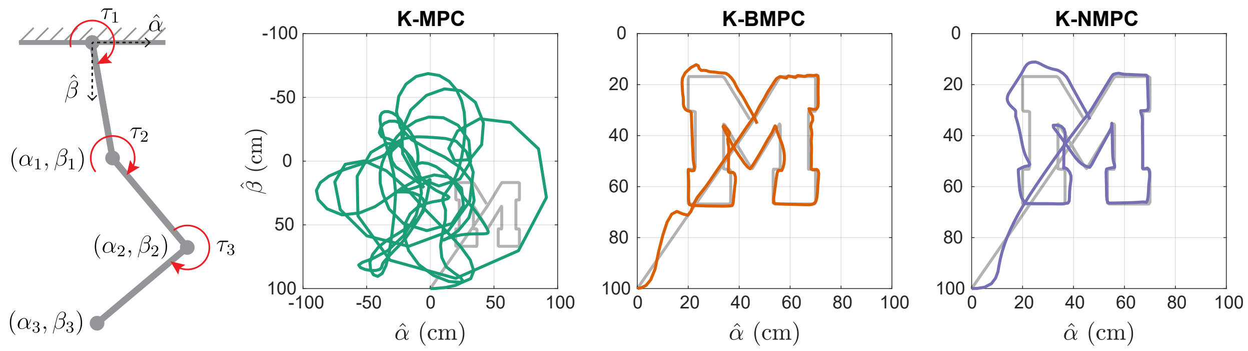

Figure 5.3: Three link planar arm system with input defined as joint torques and output defined as the locations of link tips (left). The end effector trajectories generated by the K-MPC (middle-left), K-BMPC (middle-right), and K-NMPC controllers (right) are superimposed over a reference trajectory, shown in grey.

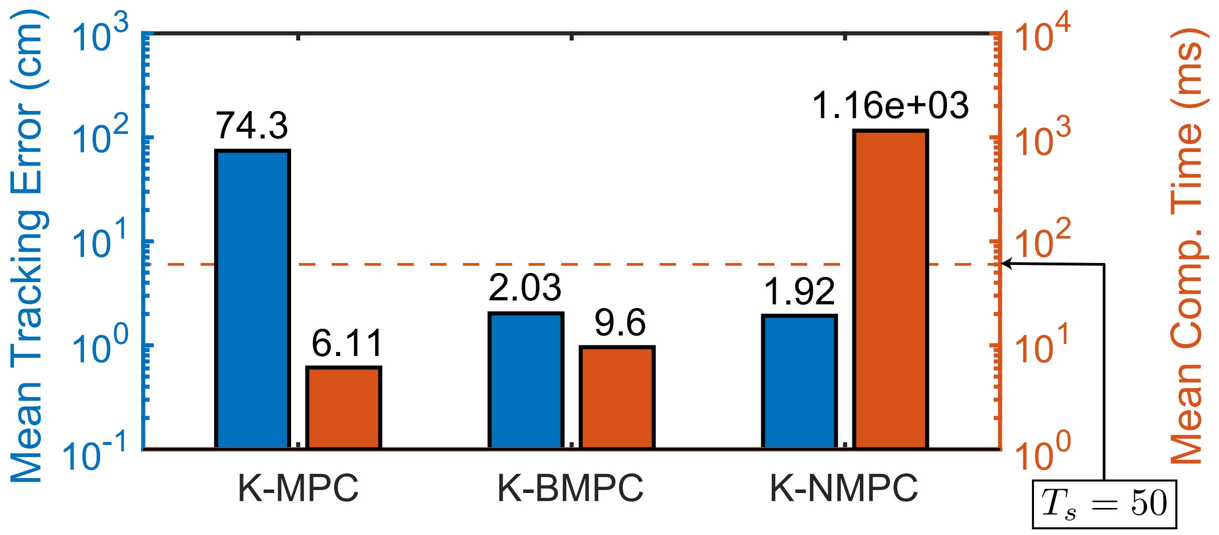

Figure 5.4: The mean tracking error (blue) and the mean computation time (orange) for each controller plotted on a logarithmic scale. The K-BMPC controller has a mean tracking error comparable to K-NMPC and a mean computation time comparable to K-MPC, proving it to be both accurate and computationally efficient.

5.4 Discussion and Conclusion

The results of the model prediction comparisons described in Section 5.3.2 illustrate the advantages of bilinear realizations for systems with unknown dynamics. As seen in in both Fig. 5.1 and Fig. 5.2, as the number of basis functions increases, the prediction error of the bilinear model realizations decreases, indicating progress toward a true infinite-dimensional bilinear realization. The linear model realizations, on the other hand, do not consistently improve with the inclusion of more basis functions, indicating that an infinite-dimensional linear realization over monomial basis functions probably doesn't exist.

This phenomenon emerges even for dynamical systems, like those governed by (5.33), which are known not to be control-affine, as shown in Fig. 5.1. The bilinear realizations outperform the linear realizations in terms of both accuracy and consistency, as indicated by their lower median error and narrower quartile range. In Section 5.2.1, we only proved the existence of infinite-dimensional bilinear realizations for control-affine systems, but these results suggest bilinear realizations may exist for a wider class of systems. This should be a topic of further study.

Bilinear Koopman realizations have benefits when it comes to control as well. It is clear by inspection of Fig. 5.3 that the K-MPC controller performs very poorly. This is confirmed quantitatively in Fig. 5.4 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 is documented in Fig. 5.2. 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 based.

In Section 5.2.3, 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. 5.4, the mean computation time for K-NMPC is more than $100\times$ 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.

This work shows that when the dynamics of a system are completely unknown, a bilinear Koopman realization constructed from data is likely to yield better overall modeling and control results than a linear or nonlinear Koopman realization. This is justified theoretically in Section 5.2.1 and demonstrated practically in Section 5.3. We proved that control-affine systems have infinite-dimensional bilinear realizations but not necessarily linear ones. Therefore, approximate bilinear realizations constructed from generic sets of basis functions improve as the number of basis functions increases, whereas approximate linear realizations may not. This enhancement directly translates to more accurate predictions for complex systems, leading to improved modeling performance. This dual benefit is particularly critical in robotics, where models ideally should be both computationally efficient to allow for real-time processing with highly accuracy in predictions in dynamic environments. Therefore, the development of bilinear Koopman realizations offers a promising advancement in the modeling and control for robotic systems.

Bilinear realizations combine the computational efficiency of linear realizations with the prediction accuracy of nonlinear realizations. However, the bilinear model predictive control framework used in this paper could likely be improved. Other techniques have been successfully applied to solve optimal control problems with bilinear constraints in [131, 79] without relying on a linear approximation at each time-step.

Further work should compare the performance of these existing bilinear optimal control approaches, investigate theoretical guarantees for bilinear controllers, and explore new approaches to optimal control of bilinear dynamical systems.

CHAPTER 6

Conclusions

6.1 Discussion of Contributions

This section briefly reviews the contributions of this document. In Chapter 2, we evaluated the effectiveness of various inertial appendages on three-dimensional body rotation through the use of physics-based simulations and trajectory optimization. The inertial appendages studied include single-link pendulum tails, articulated tails, and a variant of reaction wheels. Our results indicated that, within specific physical limitations, articulated tails were more effective than other inertial appendages in generating body rotations through the actuation of the appendages. In addition, in the part of the study where we optimized tail segment lengths, we discovered that adjusting the lengths of these segments could further improve three-dimensional body rotation. With the total tail length remaining constant, we noted a distinctive pattern in the distribution of segment lengths: segments were shorter at the tail's base and tip, with longer segments positioned in the middle. Intriguingly, this distribution mirrors that found in the mammal tails renowned for their quick and extensive body rotation abilities in comparison to mammals with less acrobatic biomechanical performance. The underlying factors contributing to the observed pattern in the length distribution observed in real animals remain an open question, prompting future research to decipher the intricacies behind the respective dynamics. The similarity between the simulation results and the biological tails likely indicates the pattern might contribute to the high maneuverability of certain animals, and may provide valuable insights for future robotic tail design, as one can consider incorporating non-uniform segments in a robotic jointed-tail.

In Chapter 3, we introduced an extensible, open-source tool based on the OpenSim API for constructing musculoskeletal models of articulated biological systems. This tool streamlines musculoskeletal model generation from scratch using just CT scan and dissection data. Furthermore, we proposed a method to replicate branched muscles and tendons more accurately. Using the tool developed, we built musculoskeletal models of artificial articulated systems with various muscle and tendon branching architectures, and compared the proposed branching modeling approach with the conventional method by conducting comparative simulations on these models. Our results showed significant differences between the models developed using the two distinct branching modeling methods, demonstrating the necessity of accurately modeling the muscle-tendon network actuation within multi-jointed biological systems. With further investigation into the impact of branching in multi-jointed biological systems, one can unravel whether there is any underlying strategy in the muscle-tendon actuation used by animals with biological continuum appendages that are crucial to their movement. The strategies employed by these systems in muscle-tendon actuation can inspire the development of more efficient and powerful actuation mechanisms in robots to mimic the coordinated movements observed in nature.

In Chapter 4, we introduced a data-driven modeling and control framework based on the Koopman operator theory. This framework transforms the representation of complex nonlinear dynamical systems into linear forms, substantially reducing the computational demands for analyzing and controlling these models as such tasks often involve optimizations based on system dynamics. We validated the efficacy of this framework by implementing it in continuum robotic systems, demonstrating its capability to linearly represent such systems. This, in turn, enables real-time closed-loop control of these continuum robotic systems. The proposed modeling and control framework thus holds significant potential for application to biological continuum appendages, which share a similar level of modeling complexity. Applying the framework to biological continuum appendages enables researchers to analyze the role of these appendages in animal movement with a significantly reduced computational burden, and even achieve real-time control of future highly articulated robotic appendages.

In Chapter 5, we introduced a data-driven framework for modeling and control that produces bilinear Koopman realizations. We provided a theoretical foundation supporting the proposed benefits of these bilinear Koopman model realizations. We define both necessary and sufficient criteria for a dynamical system to possess a valid linear or bilinear realization across a specified set of observables. Using these criteria, we demonstrate that while every control-affine system can be represented through a bilinear realization, it may not always be suitable for a linear one. Through this methodology, we illustrate that bilinear realizations provide a compromise, blending the computational efficiency associated with linear systems and the detailed representational capability of entirely nonlinear systems. Consequently, this enhances the applicability of the framework to a broader range of dynamical systems, like legged robotic systems, and the biological continuum appendages that are of particular interest in this dissertation. By enhancing the modeling accuracy while preserving computational efficiency, this method can improve the ability to achieve precise and efficient robot operation in diverse environments.

6.2 Future Directions

The ultimate aim of this work is to uncover the in-depth biological principles underlying the tail's role in the remarkable locomotive abilities observed in certain animals. This dissertation makes progress toward this goal, yet does not fully achieve it. This section highlights some of the persisting challenges and suggests possible directions for future research.

6.2.1 Musculoskeletal Modeling of Real Biological Continuum Appendages

We have developed a tool for constructing musculoskeletal models of biological articulated systems. We demonstrated its functionality by using it to build artificial musculoskeletal models. Yet, its potential in modeling detailed biological articulated systems, particularly animal tails, has not been fully realized. Constructing models that reflect real biological articulated tails and integrating real animal motion data into the models remains a unique opportunity to enhance our understanding of biomechanical strategies employed by animals in utilizing their tails.

However, constructing such models is challenging due to the need for extensive data and parameter collection essential for model development. Moreover, the models should be verified and validated to ensure the accuracy and credibility of subsequent studies. Future work should integrate CT scans and dissection data to create musculoskeletal models of real biological tails. To validate these models, researchers should collect data on tail kinematics and dynamics during contact with substrates. Investigating the tail's contribution to locomotion comprehensively requires combining these tail models with established or forthcoming legged models. By using real animal locomotion data, including both kinematics (joint positions and speeds) and dynamics (contact forces), we can gain in-depth insight into how animals exploit their tails for specific locomotor behaviors.

6.2.2 Koopman-based Data-driven Modeling and Control of Biological Continuum Appendages

We have developed a data-driven modeling and control framework based on Koopman operator theory, as presented in Chapters 4 and 5. The efficacy of this framework in modeling and controlling complex systems has been verified with real soft continuum robotic systems. These systems exhibit significant complexities, making the framework promising for application to real biological appendage motion data. The goal is to apply this framework to real biological system movement data, specifically to animal articulated tail motion data, to develop dynamical models. This could make the analysis and control of these systems more accessible while reducing computational burdens. However, the current lack of such motion data means that the application of this framework to biological systems remains unexplored. Future work should involve collecting such motion data and attempting to apply the framework for model generation.

In the meantime, it is noteworthy that the motions modeled and controlled by the Koopman-based method in real soft continuum systems may not be as dynamic as those of certain animal appendages. This arises because modeling dynamic motions typically requires a larger number of system states, more basis functions, etc., making the training of such models challenging. Recent efforts have been made to advance the Koopman-based modeling and control method from a quasi-static to an inertial, dynamic regime [55]. However, this capability relies heavily on the design of the controller. Evaluating the modeling prediction in isolation, without considering the controller in the loop, may not yield sufficiently accurate results for biological appendage cases. Therefore, future research should explore ways to improve the accuracy of modeling dynamic motions in complex systems using the Koopman-based method. This exploration is particularly interesting because the accuracy of modeling dynamic motions is crucial for understanding the connection between biological appendages and animal movement, especially since such movements are often highly dynamic.

6.2.3 Template Models Development for Continuum Appendages

To reduce the complexities involved in locomotion studies, researchers have introduced the concept of "templates" [50]. These templates represent the simplest models that capture specific behaviors, serving as guides for the control of locomotion. A widely used template model in bipedal locomotion is the spring-mass system, which consists of a massless spring attached to a point mass. This model has demonstrated its capability to characterize bipedal running across various species against empirical data. The creation of such template is grounded in more detailed models that incorporate biological details such as joints and muscles, which can be detailed musculoskeletal models of bipedal leg systems for the spring-mass model case. These detailed models are referred to as "anchor" models.

Previous investigations have tended to simply animal tails by representing them as a single rigid link to make computational analysis and control feasible. This approach offers a computationally manageable model for the complex dynamics of animal tails. However, the studies presented in this dissertation suggests that the articulated nature of tails greatly enhances their performance, in particular the inertia maneuvering performance, which indicates the need to incorporate this complexity to some degree. Therefore, a template model that simplifies tail dynamics while still addressing necessary complexities would be advantageous. Such a template model of biological tails can inform roboticists about the essential components that should be included when designing a robotic tail, which facilitates more streamlined bio-inspired design. Furthermore, given an articulated robotic tail, a template model can serve as a basis to represent the articulated robotic tail for the rapid generation of motion planning schemes and the development of control strategies. Due to the lack of detailed comprehensive anchor models, this field has been untouched. The Koopman-based method presented in this dissertation has the potential to be extended to extract governing equations that drive the dynamics of underlying systems from high-dimensional data, thereby generating a low-dimensional model. These acquired models could serve as informative templates to enhance our mechanistic understanding of tails, as well as our capability to design articulated robotic tails and their controllers for specific functions.

The ongoing development of a comprehensive musculoskeletal model for an animal tail, along with the musculoskeletal modeling tool introduced in this dissertation that streamlines the modeling process for intricate biological articulated systems, will hopefully lead to more research endeavors aimed at developing an informative template model of these complex biological continuum appendages.

6.3 Concluding Remarks

Continuum appendages, in particular animal tails, serve a multitude of functions that range from maneuverability and stability to manipulation and propulsion. Uncovering the mechanisms through which animals achieve high-performance locomotion—especially in cases involving significant tail use—remains an exciting objective for roboticists and engineers. This knowledge can aid in the development of robots capable of achieving high-performance locomotion. To reach this goal, it is crucial to thoroughly understand the fundamental principles that underlie the contribution of continuum appendages to locomotion. The tools developed in this dissertation represent steps toward achieving that goal.

BIBLIOGRAPHY

[1] Martin J . Cohn and Cheryll Tickle. Developmental basis of limblessness and axial patterning in snakes. *Nature*, 399(6735):474–479, 1999.

[2] Ian Abraham, Gerardo de la Torre, and Todd Murphey. Model-based control using koopman operators. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[3] Ian Abraham, Gerardo De La Torre, and Todd D Murphey. Model-based control using koopman operators. *arXiv preprint arXiv:1709.01568*, 2017.

[4] Ian Abraham and Todd D Murphey. Active learning of dynamics for data-driven control using koopman operators. *IEEE Transactions on Robotics*, 35(5):1071–1083, 2019.

[5] David C Ackland, Yi-Chung Lin, and Marcus G Pandy. Sensitivity of model predictions of muscle function to changes in moment arms and muscle–tendon properties: a monte-carlo analysis. *Journal of biomechanics*, 45(8):1463–1471, 2012.

[6] Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*, volume 26. Birkhäuser, 2012.

[7] John R Amend, Eric Brown, Nicholas Rodenberg, Heinrich M Jaeger, and Hod Lipson. A positive pressure universal gripper based on the jamming of granular material. *IEEE Transactions on Robotics*, 28(2):341–350, 2012.

[8] Nelly Andarawis-Puri, Eric T Ricchetti, and Louis J Soslowsky. Interaction between the supraspinatus and infraspinatus tendons: effect of anterior supraspinatus tendon full-thickness tears on infraspinatus tendon strain. *The American journal of sports medicine*, 37(9):1831–1839, 2009.

[9] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[10] Edith M Arnold, Samuel R Ward, Richard L Lieber, and Scott L Delp. A model of the lower limb for analysis of human movement. *Annals of biomedical engineering*, 38:269–279, 2010.

[11] Yuki Asano, Toyotaka Kozuki, Soichi Ookubo, Masaya Kawamura, Shinsuke Nakashima, Takeshi Katayama, Iori Yanokura, Toshinori Hirose, Kento Kawaharazuka, Shogo Makino, et al. Human mimetic musculoskeletal humanoid kengoro toward real world physically

interactive actions. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 876–883. IEEE, 2016.

[12] Henry C Astley. The biomechanics of multi-articular muscle–tendon systems in snakes. *Integrative and Comparative Biology*, 60(1):140–155, 2020.

[13] Karl Johan Aström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[14] Craig Bakker, Steven Rosenthal, and Kathleen E Nowak. Koopman representations of dynamic systems with control. *arXiv preprint arXiv:1908.02233*, 2019.

[15] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[16] Peter J Bishop, Andrew R Cuff, and John R Hutchinson. How to build a dinosaur: Musculoskeletal modeling and simulation of locomotor biomechanics in extinct animals. *Paleobiology*, 47(1):1–38, 2021.

[17] Peter J Bishop, Antoine Falisse, Friedl De Groote, and John R Hutchinson. Predictive simulations of running gait reveal a critical dynamic role for the tail in bipedal dinosaur locomotion. *Science advances*, 7(39):eabi7348, 2021.

[18] Peter J Bishop, Krijn B Michel, Antoine Falisse, Andrew R Cuff, Vivian R Allen, Friedl De Groote, and John R Hutchinson. Computational modelling of muscle fibre operating ranges in the hindlimb of a small ground bird (eudromia elegans), with implications for modelling locomotion in extinct species. *PLoS Computational Biology*, 17(4):e1008843, 2021.

[19] Yoann Blache, Benjamin Michaud, Isabelle Rogowski, K Monteil, and Mickaël Begon. Sensitivity of shoulder musculoskeletal model predictions to muscle–tendon properties. *IEEE Transactions on Biomedical Engineering*, 66(5):1309–1317, 2018.

[20] David B Boerma, Kenneth S Breuer, Tim L Treskatis, and Sharon M Swartz. Wings as inertial appendages: how bats recover from aerial stumbles. *Journal of Experimental Biology*, 222(20):jeb204255, 2019.

[21] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[22] Randall Briggs, Jongwoo Lee, Matt Haberland, and Sangbae Kim. Tails in biomimetic design: Analysis, simulation, and experiment. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 1473–1480. IEEE, 2012.

[23] Daniel Bruder, Xun Fu, R Brent Gillespie, C David Remy, and Ram Vasudevan. Koopman-based control of a soft continuum manipulator under variable loading conditions. *arXiv preprint arXiv:2002.01407*, 2020.

[24] Daniel Bruder, Brent Gillespie, C. David Remy, and Ram Vasudevan. Modeling and control of soft robots using the koopman operator and model predictive control. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[25] Daniel Bruder, C David Remy, and Ram Vasudevan. Nonlinear system identification of soft robot dynamics using koopman operator theory. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6244–6250. IEEE, 2019.

[26] Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PloS one*, 11(2), 2016.

[27] Marko Budišić, Ryan Mohr, and Igor Mezić. Applied koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[28] P Bujalski, J Martins, and L Stirling. A monte carlo analysis of muscle force estimation sensitivity to muscle-tendon properties using a hill-based muscle model. *Journal of biomechanics*, 79:67–77, 2018.

[29] Malcolm Burrows, Darron A Cullen, Marina Dorosenko, and Gregory P Sutton. Mantises exchange angular momentum between three rotating body parts to jump precisely to targets. *Current Biology*, 25(6):786–789, 2015.

[30] Aurelio Cappozzo, Fabio Catani, Ugo Della Croce, and Alberto Leardini. Position and orientation in space of bones during movement: anatomical frame definition and determination. *Clinical biomechanics*, 10(4):171–178, 1995.

[31] Vincenzo Carbone, MM Van der Krogt, Hubertus FJM Koopman, and Nico Verdonschot. Sensitivity of subject-specific models to hill muscle–tendon model parameters in simulations of gait. *Journal of biomechanics*, 49(9):1953–1960, 2016.

[32] Carlos Casarez, Ivan Penskiy, and Sarah Bergbreiter. Using an inertial tail for rapid turns on a miniature legged robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 5469–5474. IEEE, 2013.

[33] LINDSEY CC. Form, function, and locomotory habits in fish. *Fish physiology*, pages 1–100, 1978.

[34] Evan Chang-Siu, Thomas Libby, Masayoshi Tomizuka, and Robert J Full. A lizard-inspired active tail enables rapid maneuvers and dynamic stabilization in a terrestrial robot. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1887–1894. IEEE, 2011.

[35] James P Charles, Ornella Cappellari, and John R Hutchinson. A dynamic simulation of musculoskeletal function in the mouse hindlimb during trotting locomotion. *Frontiers in Bioengineering and Biotechnology*, 6:61, 2018.

[36] Liang Chen, Ziang Jiang, Chen Yang, Rongshan Cheng, Size Zheng, and Jingguang Qian. Effect of different landing actions on knee joint biomechanics of female college athletes: Based on opensim simulation. *Frontiers in Bioengineering and Biotechnology*, 10:899799, 2022.

[37] Xiangyu Chu, Shengzhi Wang, Raymond Ng, Chun Yin Fan, Jiajun An, and KW Samuel Au. Combining tail and reaction wheel for underactuated spatial reorientation in robot falling with quadratic programming. *IEEE Robotics and Automation Letters*, 2023.

[38] SM Cox, KL Easton, M Cromie Lear, RL Marsh, SL Delp, and J Rubenson. The interaction of compliance and activation on the force-length operating range and force generating capacity of skeletal muscle: a computational study using a guinea fowl musculoskeletal model. *Integrative Organismal Biology*, 1(1):obz022, 2019.

[39] Friedl De Groote and Antoine Falisse. Perspective on musculoskeletal modelling and predictive simulations of human movement to assess the neuromechanics of gait. *Proceedings of the Royal Society B*, 288(1946):20202432, 2021.

[40] Friedl De Groote, Anke Van Campen, Ilse Jonkers, and Joris De Schutter. Sensitivity of dynamic simulations of gait and dynamometer experiments to hill muscle model parameters of knee flexors and extensors. *Journal of biomechanics*, 43(10):1876–1883, 2010.

[41] Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*, 54(11):1940–1950, 2007.

[42] Scott L Delp and J Peter Loan. A graphics-based software system to develop and analyze models of musculoskeletal structures. *Computers in biology and medicine*, 25(1):21–34, 1995.

[43] Scott L Delp and J Peter Loan. A computational framework for simulating and analyzing human and animal movement. *Computing in Science & Engineering*, 2(5):46–55, 2000.

[44] Scott L Delp, J Peter Loan, Melissa G Hoy, Felix E Zajac, Eric L Topp, and Joseph M Rosen. An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. *IEEE Transactions on Biomedical engineering*, 37(8):757–767, 1990.

[45] Michael H Dickinson, Claire T Farley, Robert J Full, MAR Koehl, Rodger Kram, and Steven Lehman. How animals move: an integrative view. *science*, 288(5463):100–106, 2000.

[46] Antoine Falisse, Gil Serrancolí, Christopher L Dembia, Joris Gillis, Ilse Jonkers, and Friedl De Groote. Rapid predictive simulations with complex musculoskeletal models suggest that diverse healthy and pathological human gaits can emerge from similar control strategies. *Journal of The Royal Society Interface*, 16(157):20190402, 2019.

[47] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[48] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, et al. 3d slicer as an image computing platform for the quantitative imaging network. *Magnetic resonance imaging*, 30(9):1323–1341, 2012.

[49] Benjamin J Fregly, Jeffrey A Reinbolt, Kelly L Rooney, Kim H Mitchell, and Terese L Chmielewski. Design of patient-specific gait modifications for knee osteoarthritis rehabilitation. *IEEE Transactions on Biomedical Engineering*, 54(9):1687–1695, 2007.

[50] Robert J Full and Daniel E Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of experimental biology*, 202(23):3325–3332, 1999.

[51] Roger V Gonzalez, Thomas S Buchanan, and Scott L Delp. How muscle architecture and moment arms affect wrist flexion-extension moments. *Journal of biomechanics*, 30(7):705–712, 1997.

[52] Debdipta Goswami and Derek A Paley. Global bilinearization and reachability analysis of control-affine nonlinear systems. In *The Koopman Operator in Systems and Control*, pages 81–98. Springer, 2020.

[53] Sten Grillner and Abdeljabbar El Manira. Current principles of motor control, with special reference to vertebrate locomotion. *Physiological reviews*, 2019.

[54] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[55] David A Haggerty, Michael J Banks, Ervin Kamenar, Alan B Cao, Patrick C Curtis, Igor Mezić, and Elliot W Hawkes. Control of soft robots with inertial dynamics. *Science robotics*, 8(81):eadd6864, 2023.

[56] Ayonga Hereid and Aaron D Ames. Frost: Fast robot optimization and simulation toolkit. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 719–726. IEEE, 2017.

[57] Graham C Hickman. The mammalian tail: a review of functions. *Mammal review*, 9(4):143–157, 1979.

[58] Nicholas J Higham. *Functions of matrices: theory and computation*, volume 104. Siam, 2008.

[59] Hoa X Hoang and Jeffrey A Reinbolt. Crouched posture maximizes ground reaction forces generated by muscles. *Gait & posture*, 36(3):405–408, 2012.

[60] Katherine RS Holzbaur, Wendy M Murray, and Scott L Delp. A model of the upper extremity for simulating musculoskeletal surgery and analyzing neuromuscular control. *Annals of biomedical engineering*, 33:829–840, 2005.

[61] Huang Huang, Antonio Loquercio, Ashish Kumar, Neerja Thakkar, Ken Goldberg, and Jitendra Malik. More than an arm: Using a manipulator as a tail for enhanced stability in legged locomotion. *arXiv preprint arXiv:2305.01648*, 2023.

[62] John R Hutchinson, Jeffery W Rankin, Jonas Rubenson, Kate H Rosenbluth, Robert A Siston, and Scott L Delp. Musculoskeletal modelling of an ostrich (struthio camelus) pelvic limb: influence of limb orientation on muscular capacity during locomotion. *PeerJ*, 3:e1001, 2015.

[63] Auke J Ijspeert. Amphibious and sprawling locomotion: from biology to robotics and back. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:173–193, 2020.

[64] Ardian Jusufi, Daniel I Goldman, Shai Revzen, and Robert J Full. Active tails enhance arboreal acrobatics in geckos. *Proceedings of the National Academy of Sciences*, 105(11):4215–4219, 2008.

[65] Ardian Jusufi, Daniel T Kawano, Thomas Libby, and Robert J Full. Righting and turning in mid-air using appendage inertia: reptile tails, analytical models and bio-inspired robots. *Bioinspiration & biomimetics*, 5(4):045001, 2010.

[66] Tom Kalisky, Yueqi Wang, Benjamin Shih, Dylan Drotman, Saurabh Jadhav, Eliah Aronoff-Spencer, and Michael T Tolley. Differential pressure control of 3d printed soft fluidic actuators. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6207–6213. IEEE, 2017.

[67] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[68] Seth A. Kewley A, Beesel J. Opensim creator. *Zenodo*, 2023.

[69] Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018.

[70] Arthur D Kuo and J Maxwell Donelan. Dynamic principles of gait and their clinical implications. *Physical therapy*, 90(2):157–174, 2010.

[71] Andrzej Lasota and Michael C Mackey. *Chaos, fractals, and noise: stochastic aspects of dynamics*, volume 97. Springer Science & Business Media, 2013.

[72] Viet Le, Benjamin Cellini, Rudolf Schilder, and Jean-Michel Mongeau. Hawkmoths regulate flight torques with their abdomen for yaw control. *Journal of Experimental Biology*, 226(9):jeb245063, 2023.

[73] Thomas Libby, Talia Y Moore, Evan Chang-Siu, Deborah Li, Daniel J Cohen, Ardian Jusufi, and Robert J Full. Tail-assisted pitch control in lizards, robots and dinosaurs. *Nature*, 481(7380):181–184, 2012.

[74] Richard L Lieber, Mark D Jacobson, Babak M Fazeli, Reid A Abrams, and Michael J Botte. Architecture of selected muscles of the arm and forearm: anatomy and implications for tendon transfer. *The Journal of hand surgery*, 17(5):787–798, 1992.

[75] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 121–131, 2007.

[76] May Q Liu, Frank C Anderson, Michael H Schwartz, and Scott L Delp. Muscle contributions to support and progression over a range of walking speeds. *Journal of biomechanics*, 41(15):3243–3252, 2008.

[77] Yujiong Liu and Pinhas Ben-Tzvi. Design, analysis, and integration of a new two-degree-of-freedom articulated multi-link robotic tail mechanism. *Journal of Mechanisms and Robotics*, 12(2):021101, 2020.

[78] David G Lloyd, Ilse Jonkers, Scott L Delp, and Luca Modenese. The history and future of neuromusculoskeletal biomechanics. *Journal of Applied Biomechanics*, 39(5):273–283, 2023.

[79] Xu Ma, Bowen Huang, and Umesh Vaidya. Optimal quadratic regulation of nonlinear system using koopman operator. In *2019 American Control Conference (ACC)*, pages 4911–4916. IEEE, 2019.

[80] Alexander R MacIntosh and Peter J Keir. An open-source model and solution method to predict co-contraction in the finger. *Computer Methods in Biomechanics and Biomedical Engineering*, 20(13):1373–1381, 2017.

[81] Srikrishnaraja Mahadas, Kausalendra Mahadas, and George K Hung. Biomechanics of the golf swing using opensim. *Computers in biology and medicine*, 105:39–45, 2019.

[82] Giorgos Mamakoukas, Maria Castano, Xiaobo Tan, and Todd Murphey. Local koopman operators for data-driven control of robotic systems. In *Robotics: science and systems*, 2019.

[83] MATLAB. *version 7.10.0 (R2017a)*. The MathWorks Inc., Natick, Massachusetts, 2017.

[84] Alexandre Mauroy and Jorge Goncalves. Linear identification of nonlinear systems: A lifting technique based on the koopman operator. *arXiv preprint arXiv:1605.04457*, 2016.

[85] Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *arXiv preprint arXiv:1709.02003*, 2017.

[86] Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *IEEE Transactions on Automatic Control*, 2019.

[87] Daniel C McFarland, Benjamin I Binder-Markey, Jennifer A Nichols, Sarah J Wohlman, Marije de Bruin, and Wendy M Murray. A musculoskeletal model of the hand and wrist capable of simulating functional tasks. *IEEE Transactions on Biomedical Engineering*, 70(5):1424–1435, 2022.

[88] Josh Merel, Matthew Botvinick, and Greg Wayne. Hierarchical motor control in mammals and machines. *Nature communications*, 10(1):5489, 2019.

[89] Matthew Millard, Thomas Uchida, Ajay Seth, and Scott L Delp. Flexing computational muscle: modeling and simulation of musculotendon dynamics. *Journal of biomechanical engineering*, 135(2):021005, 2013.

[90] R Matthew Miller, James Thunes, Volker Musahl, Spandan Maiti, and Richard E Debski. Effects of tear size and location on predictions of supraspinatus tear propagation. *Journal of biomechanics*, 68:51–57, 2018.

[91] Moore T.Y. Miyamae J.A. Entails a closer look: Comparative muscular morphology and function of the mammalian tail. *Society for Integrative and Comparative Biology*, 2023.

[92] Moore T.Y. Miyamae J.A. Telling tails: Comparative muscular morphology and function of mammalian tails. *International Congress of Vertebrate Morphology*, 2023.

[93] Brad R Moon and Carl Gans. Kinematics, muscular activity and propulsion in gopher snakes. *Journal of Experimental Biology*, 201(19):2669–2684, 1998.

[94] Eduardo Martin Moraud, Marco Capogrosso, Emanuele Formento, Nikolaus Wenger, Jack DiGiovanna, Gregoire Courtine, and Silvestro Micera. Mechanisms underlying the neuromodulation of spinal circuits for correcting gait and balance deficits after spinal cord injury. *Neuron*, 89(4):814–828, 2016.

[95] Richard R Neptune, Steven A Kautz, and Felix E Zajac. Contributions of the individual ankle plantar flexors to support, forward progression and swing initiation during walking. *Journal of biomechanics*, 34(11):1387–1398, 2001.

[96] Joseph Norby, Jun Yang Li, Cameron Selby, Amir Patel, and Aaron M Johnson. Enabling dynamic behaviors with aerodynamic drag in lightweight tails. *IEEE Transactions on Robotics*, 37(4):1144–1153, 2021.

[97] Shawn M O'Connor, Terence J Dawson, Rodger Kram, and J Maxwell Donelan. The kangaroo's tail propels and powers pentapedal locomotion. *Biology letters*, 10(7):20140381, 2014.

[98] Matthew C O'Neill, Leng-Feng Lee, Susan G Larson, Brigitte Demes, Jack T Stern Jr, and Brian R Umberger. A three-dimensional musculoskeletal model of the chimpanzee (pan troglodytes) pelvis and hind limb. *Journal of Experimental Biology*, 216(19):3709–3723, 2013.

[99] Amir Patel and M Braae. Rapid acceleration and braking: Inspirations from the cheetah's tail. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 793–799. IEEE, 2014.

[100] Amir Patel and Martin Braae. Rapid turning at high-speed: Inspirations from the cheetah's tail. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5506–5511. IEEE, 2013.

[101] Michael A Patterson and Anil V Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1, 2014.

[102] Joel A Paulson, Ali Mesbah, Stefan Streif, Rolf Findeisen, and Richard D Braatz. Fast stochastic model predictive control of high-dimensional systems. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 2802–2809. IEEE, 2014.

[103] Sebastian Peitz, Samuel E Otto, and Clarence W Rowley. Data-driven model predictive control using interpolated koopman generators. *SIAM Journal of Applied Dynamical Systems*, 19(3), 2020.

[104] Roger Penrose. On best approximate solutions of linear matrix equations. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 52, pages 17–19. Cambridge University Press, 1956.

[105] Harold Perkel. Stabilite–a three-axis attitude control system utilizing a single reaction wheel. In *Progress in Astronautics and Rocketry*, volume 19, pages 375–400. Elsevier, 1966.

[106] Elijah Polak. *Optimization: algorithms and consistent approximations*, volume 124. Springer Science & Business Media, 2012.

[107] Isaac Pressgrove, Yujiong Liu, and Pinhas Ben-Tzvi. Design and implementation of a power-dense, modular, and compact serpentine articulated tail. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 86281, page V007T07A053. American Society of Mechanical Engineers, 2022.

[108] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

[109] Andrew O Pullin, Nicholas J Kohut, David Zarrouk, and Ronald S Fearing. Dynamic turning of 13 cm robot comparing tail and differential drive. In *2012 IEEE international conference on robotics and automation*, pages 5086–5093. IEEE, 2012.

[110] Apoorva Rajagopal, Christopher L Dembia, Matthew S DeMers, Denny D Delp, Jennifer L Hicks, and Scott L Delp. Full-body musculoskeletal model for muscle-driven simulation of human gait. *IEEE transactions on biomedical engineering*, 63(10):2068–2079, 2016.

[111] Shravan Tata Ramalingasetty, Simon M Danner, Jonathan Arreguit, Sergey N Markin, Dimitri Rodarie, Claudia Kathe, Grégoire Courtine, Ilya A Rybak, and Auke Jan Ijspeert. A whole-body musculoskeletal model of the mouse. *Ieee Access*, 9:163861–163881, 2021.

[112] John Rasmussen, Michael Damsgaard, Egidijus Surma, Søren T Christensen, Mark De Zee, and Vit Vondrak. Anybody-a software system for ergonomic optimization. In *Fifth world congress on structural and multidisciplinary optimization*, volume 4, 2003.

[113] James B Rawlings and David Q Mayne. *Model predictive control: Theory and design*. Nob Hill Pub. Madison, Wisconsin, 2009.

[114] Christian Redl, Margit Gfoehler, and Marcus G Pandy. Sensitivity of muscle force estimates to variations in muscle–tendon properties. *Human movement science*, 26(2):306–319, 2007.

[115] Bernard J Rigby, Nishio Hirai, John D Spikes, and Henry Eyring. The mechanical properties of rat tail tendon. *The Journal of general physiology*, 43(2):265–283, 1959.

[116] William Rone and Pinhas Ben-Tzvi. Dynamic modeling and simulation of a yaw-angle quadruped maneuvering with a planar robotic tail. *Journal of Dynamic Systems, Measurement, and Control*, 138(8):084502, 2016.

[117] Paulien E Roos, Anita Vasavada, Liying Zheng, and Xianlian Zhou. Neck musculoskeletal model generation through anthropometric scaling. *Plos one*, 15(1):e0219954, 2020.

[118] Andre Rosendo, Shogo Nakatsu, Kenichi Narioka, and Koh Hosoda. Pneupard: A biomimetic musculoskeletal approach for a feline-inspired quadruped robot. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1452–1457. IEEE, 2013.

[119] Wael Saab, William S Rone, Anil Kumar, and Pinhas Ben-Tzvi. Design and integration of a novel spatial articulated robotic tail. *IEEE/ASME Transactions on Mechatronics*, 24(2):434–446, 2019.

[120] Sreeshankar Satheeshbabu, Naveen Kumar Uppalapati, Girish Chowdhary, and Girish Krishnan. Open loop position control of soft continuum arm using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5133–5139. IEEE, 2019.

[121] Stefan Schmid, Katelyn A Burkhart, Brett T Allaire, Daniel Grindle, and Dennis E Anderson. Musculoskeletal full-body models including a detailed thoracolumbar spine for children and adolescents aged 6–18 years. *Journal of biomechanics*, 102:109305, 2020.

[122] Carol Y Scovil and Janet L Ronsky. Sensitivity of a hill-based muscle model to perturbations in model parameters. *Journal of biomechanics*, 39(11):2055–2063, 2006.

[123] Ajay Seth, Jennifer L Hicks, Thomas K Uchida, Ayman Habib, Christopher L Dembia, James J Dunne, Carmichael F Ong, Matthew S DeMers, Apoorva Rajagopal, Matthew Millard, et al. Opensim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLoS computational biology*, 14(7):e1006223, 2018.

[124] Maziar Ahmad Sharbafi, Christian Rode, Stefan Kurowski, Dorian Scholz, Rico Möckel, Katayon Radkhah, Guoping Zhao, Aida Mohammadinejad Rashty, Oskar von Stryk, and Andre Seyfarth. A new biarticular actuator design facilitates control of leg function in biobiped3. *Bioinspiration & biomimetics*, 11(4):046003, 2016.

[125] Harumichi SHINOHARA. The mouse vertebrae: changes in the morphology of mouse vertebrae exhibit specific patterns over limited numbers of vertebral levels. *Okajimas folia anatomica japonica*, 76(1):17–31, 1999.

[126] Avinash Singh, Thomas Libby, and Sawyer B Fuller. Rapid inertial reorientation of an aerial insect-sized robot using a piezo-actuated tail. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4154–4160. IEEE, 2019.

[127] Heiko Stark, Martin S Fischer, Alexander Hunt, Fletcher Young, Roger Quinn, and Emanuel Andrada. A three-dimensional musculoskeletal model of the dog. *Scientific reports*, 11(1):11335, 2021.

[128] Katherine M Steele, Adam Rozumalski, and Michael H Schwartz. Muscle synergies and complexity of neuromuscular control during gait in cerebral palsy. *Developmental Medicine & Child Neurology*, 57(12):1176–1182, 2015.

[129] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. In *2018 UKACC 12th International Conference on Control (CONTROL)*, pages 339–339. IEEE, 2018.

[130] Justin Storms and Dawn Tilbury. Dynamic weight-shifting for improved maneuverability and rollover prevention in high-speed mobile manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 138(10):101007, 2016.

[131] Amit Surana. Koopman operator based observer synthesis for control-affine nonlinear systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6492–6499. IEEE, 2016.

[132] Bertrand Tondu. Modelling of the mckibben artificial muscle: A review. *Journal of Intelligent Material Systems and Structures*, 23(3):225–253, 2012.

[133] Scott D Uhlrich, Thomas K Uchida, Marissa R Lee, and Scott L Delp. Ten steps to becoming a musculoskeletal simulation expert: A half-century of progress and outlook for the future. *Journal of Biomechanics*, page 111623, 2023.

[134] Giordano Valente, Gianluigi Crimi, Nicola Vanella, Enrico Schileo, and Fulvia Taddei. nmsbuilder: Freeware to create subject-specific musculoskeletal models for opensim. *Computer methods and programs in biomedicine*, 152:85–92, 2017.

[135] Pasha A Van Bijlert, AJ 'Knoek' van Soest, and Anne S Schulp. Natural frequency method: estimating the preferred walking speed of tyrannosaurus rex based on tail natural frequency. *Royal Society Open Science*, 8(4):201441, 2021.

[136] Peter Van Overschee and BL De Moor. *Subspace identification for linear systems: Theory—Implementation—Applications*. Springer Science & Business Media, 2012.

[137] Kunyang Wang, Lei Ren, Zhihui Qian, Jing Liu, Tao Geng, and Luquan Ren. Development of a 3d printed bipedal robot: towards humanoid research platform to study human musculoskeletal biomechanics. *Journal of Bionic Engineering*, 18:150–170, 2021.

[138] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

[139] Alan M Wilson, JC Lowe, K Roskilly, Penny E Hudson, KA Golabek, and JW McNutt. Locomotion dynamics of hunting in wild cheetahs. *Nature*, 498(7453):185–189, 2013.

[140] Ge Wu, Sorin Siegler, Paul Allard, Chris Kirtley, Alberto Leardini, Dieter Rosenbaum, Mike Whittle, Darryl D D'Lima, Luca Cristofolini, Hartmut Witte, et al. Isb recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion—part i: ankle, hip, and spine. *Journal of biomechanics*, 35(4):543–548, 2002.

[141] Ge Wu, Frans CT Van der Helm, HEJ DirkJan Veeger, Mohsen Makhsous, Peter Van Roy, Carolyn Anglin, Jochem Nagels, Andrew R Karduna, Kevin McQuade, Xuguang Wang, et al. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion—part ii: shoulder, elbow, wrist and hand. *Journal of biomechanics*, 38(5):981–992, 2005.

[142] Ming Xiao and Jill Higginson. Sensitivity of estimated muscle force in forward simulation of normal walking. *Journal of applied biomechanics*, 26(2):142–149, 2010.

[143] Yasunori Yamada, Satoshi Nishikawa, Kazuya Shida, Ryuma Niiyama, and Yasuo Kuniyoshi. Neural-body coupling for emergent locomotion: A musculoskeletal quadruped robot with spinobulbar model. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1499–1506. IEEE, 2011.

[144] Yanhao Yang, Joseph Norby, Justin K Yim, and Aaron M Johnson. Proprioception and tail control enable extreme terrain traversal by quadruped robots. *arXiv preprint arXiv:2303.04781*, 2023.

[145] Fletcher Young, Christian Rode, Alex Hunt, and Roger Quinn. Analyzing moment arm profiles in a full-muscle rat hindlimb model. *Biomimetics*, 4(1):10, 2019.

[146] Paul A Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C Gee, and Guido Gerig. User-guided 3d active contour segmentation of anatomical structures: significantly improved efficiency and reliability. *Neuroimage*, 31(3):1116–1128, 2006.

[147] Zhiqiang Zhang, Jialing Yang, and Hui Yu. Effect of flexible back on energy absorption during landing in cats: a biomechanical investigation. *Journal of Bionic Engineering*, 11(4):506–516, 2014.

[148] Jianguo Zhao, Tianyu Zhao, Ning Xi, Matt W Mutka, and Li Xiao. Msu tailbot: Controlling aerial maneuver of a miniature-tailed jumping robot. *IEEE/ASME Transactions on Mechatronics*, 20(6):2903–2914, 2015.

[149] Pengcheng Zhao, Shankar Mohan, and Ram Vasudevan. Control synthesis for nonlinear optimal control via convex relaxations. In *American Control Conference (ACC), 2017*, pages 2654–2661. IEEE, 2017.