Applications of the Koopman Operator: Novel Methods and Formulations for Lifted Linear Models

by

Jerry Ng

B.S., University of California - San Diego (2017)

S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Author .....

Department of Mechanical Engineering

February 22, 2023

Certified by .....

H. Harry Asada

Ford Professor of Engineering

Thesis Supervisor

Accepted by .....

Nicolas Hadjiconstantinou

Professor of Mechanical Engineering

Chairman, Department Committee on Graduate Theses

Applications of the Koopman Operator:

Novel Methods and Formulations for Lifted Linear Models

by

Jerry Ng

Submitted to the Department of Mechanical Engineering
on February 22, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Abstract

The analysis and control of nonlinear dynamic systems is an active research field due to the ubiquity of nonlinear systems in the physical world. However, the handling of these systems is significantly more difficult than the handling of their linear counterparts, for which a host of methods and techniques are available. It has been shown that through the use of the Koopman Operator, these nonlinear systems can be lifted to a higher order state space, and with this lifted representation, the system's dynamics behave linearly.

In this thesis, we explore the use of existing methods for constructing the Koopman Operator on unexplored classes of nonlinear systems, such as systems with segmented dynamics and exogenous inputs. Unlike when modeling these systems with a hybrid or switched framework, the lifted linear models based on the Koopman Operator allow for easy application of model predictive control.

We then discuss the methods for constructing the Koopman Operator. Specifically, we alleviate the pitfalls of current data-driven methods for construction of the Koopman Operator through the use of a data-driven formulation of Direct Encoding, which is based on integration. This differs significantly from the state of the art.

Lastly, the use of Koopman with relation to deep learning is considered. Through utilizing the aforementioned data-driven method, improvements to standard applications of Deep Koopman are demonstrated. In addition, we demonstrate a novel training method that is enabled by Direct Encoding. Through the use of this method, we are able to accurately model the stable subspace of a system containing both stable and unstable subspaces, unlike with standard Deep Koopman methods. It is shown that the resultant model can be used to estimate the borders between subspaces.

Thesis Supervisor: H. Harry Asada
Title: Ford Professor of Engineering

Acknowledgments

Naturally, the first person I would like to acknowledge is my advisor, Professor Harry H. Asada for his guidance, teaching, and mentoring during my studies at MIT. Working alongside him has been instrumental in my growth both academically and as an individual.

I would also like to thank the members of my committee, Professor Neville Hogan and Doctor Anuradha Annaswamy for their insights, perspectives, and knowledge they've shared.

I would like to express my gratitude towards my sponsors throughout my doctoral study, MathWorks and the National Science Foundation.

Of course, I am thankful to my colleagues and friends within the d'Arbeloff Lab that have made my time at MIT bearable and, though rarely, pleasant. While everyone has impacted me significantly through my experiences with you, there are a few individuals that I would be ashamed if I did not highlight. The first person is Jacob Guggenheim for being a primary driver in getting me to join the lab in the first place. You have been a mentor and my voice of reason. The next person I need to highlight is Nick Selby. While Jacob was my voice of reason, I owe the development of my moral compass to Nick and his ability to push forward with an unwavering desire to do good for the world while making no exceptions for himself. To the golden boy, Filippos Sotiropoulos, working alongside you gave me an image for what healthy productivity and an understanding of what it means to be reasonable. And then there's Abbas. It is through our conversations, though the frequency has decreased, that I get a better understanding of the world around me from a perspective that I rarely encounter. There are of course others that I wish to highlight, but this acknowledgements section is getting quite long. So to Zoe, Ryan, John Bell, Phillip, Rachel, Katie, Cormac, Emily, and all of the others that I have not mentioned, I simply don't have the brain power in me to conjure a unique enough sentence that is not a derivative of the aforementioned thank yous and this amalgamation will have to suffice.

To my friends and family outside of MIT, I appreciate the support you have given me and hope that I will be able to return that support in kind. I owe a particularly special thank you to my close friend, Julia Ting. You are the person that I've talked to the most, complained to the most, and the person whom I've trusted with my most deep seated insecurities, fears, and anxieties throughout the years. You have managed to keep me sane, and for that I am eternally grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The primary concern of this work regards creation and utilization of linearizations of nonlinear systems through data-driven approaches. Nonlinear systems are ubiquitous in the world, but difficult to handle. Though a collection of tools to address these systems have been created, there still exist classes of nonlinear systems that are difficult to address with current tools. When attempting to control, model, or even analyze such systems, one must go to far greater lengths than when controlling or analyzing a benign linear dynamic system. Considering this comparison, the desire to model and analyze these nonlinear systems with linear approximations is not novel.

What is new for the world is the vast and abundant computing and memory resources that are accessible. With these resources, applications of older theory become far more tractable. The primary theory that this work draws from is a paper written in 1931 by Bernard O. Koopman, in which Koopman discussed that a nonlinear dynamic system could be described by a spectrum of characterizing functions in a linear fashion [23].

Dynamic Mode Decomposition (DMD) was presented as a method to produce linear models from sequential data generated through nonlinear dynamical processes by using Singular Value Decomposition (SVD) [44]. This method enabled analysis of high dimensional nonlinear systems through separating its dynamics into simpler linear modes, hence the modal decomposition-based name. The analysis hinged upon a linear model which allowed for the superposition of the linear modes to describe

the more complex nonlinear system. Later, DMD was developed further to create Extended Dynamic Mode Decomposition (EDMD), which introduced the concept of using observable functions, nonlinear functions of state variables, as a method of augmenting the state space [50]. It was in this work that references were drawn to the Koopman Operator, providing justification and a theoretical underpinning for lifting the state space. These methods were initially applied to fluid flows, autonomous systems that exhibit significant nonlinear dynamics, and these applications were successful in describing and providing understanding of the nonlinear systems. Further work extended the use cases of DMD, demonstrating viability in providing linear models of nonlinear dynamic systems even in the presence of exogenous inputs in the case of non-autonomous systems [43]. This method, DMDc, was capable of creating a model that could be used for control. This enabled complex nonlinear Model Predictive Control (MPC) to be converted to linear MPC [26], leading to numerous studies that applied the methodology to real systems.

From these initial works regarding DMD, three main avenues of research formed regarding the Koopman Operator. The first was in application of the method. Though the original applications were regarding fluids, after the method was extended to allow for control, novel applications included: including: vehicles [12], soft robotics [7], gait recognition [53], and bucket-soil interactions from excavation [48]. On the frontier of applications are hybrid systems; these systems are unable to be defined by a single dynamic equation and instead are defined with multiple dynamic equations that are used depending on the state. The second avenue of research regards the selection of observable functions uses for constructing a lifted state space, as these functions are a key ingredient in creating an accurate linear model. Various methods have been developed, including deep neural networks for learning effective observable functions [33, 46, 51] and optimization [27]. However, an efficient selection of observables does not solve all the issues that arise when attempting to construct an accurate linear model. For example, it is known that unstable modes form when utilizing Koopman-based DMD models and their extensions, even in cases where the underlying nonlinear systems are known to be stable; extensive studies have been done to create stable

linear models to remedy the situations where an outright use of DMD would lead to the creation of an unstable linear model [6, 15, 18, 35]. These studies constitute the third avenue of research: alternative formulations of the linear transition matrix. This linear transition matrix is the final output of the Koopman Operator framework as it describes the dynamics of the nonlinear system in the lifted linear model. This thesis consists of contributions to each of these three avenues of research.

With regards to applications, this thesis addresses the application of the Koopman Operator to hybrid systems that pose problematic for modern robotics research. In relation to the second avenue of selection of observable functions, we present novel training methods for Deep Learning to produce observable functions from datasets. Along the third avenue of research in the Koopman Operator, we present a numerical integration-based method that serves as a method of creating a linear model that is robust to biases in distributions of datasets.

1.1 Original Contributions

The contributions presented in this work are as follows:

1. An approximation of the Koopman Operator is applied to hybrid dynamic systems, and the model is used for both control and prediction of different but similar systems (Chapter 3).

2. A method is constructed to robustly construct approximations of the Koopman Operator from data, utilizing a numerical integration approach opposed to the state-of-the-art Least Squares Estimation based approaches (Chapter 4).

3. A method is constructed to generate subspace specific observable functions using deep learning structures to accurately predict stable subspaces in the presence of data belonging to unstable subspaces (Chapter 5).

1.2 Thesis Structure

Chapter 2 introduces the formulation of the Koopman Operator, first with the Least Squares Estimation approach and then with a direct encoding using the nonlinear mapping. Chapter 3 discusses the use of existing lifted linearization methods on complex nonlinear systems, and the families of nonlinear systems that pose possible problems for application of the technique. Constituting the first contribution, the application of the Koopman Operator to switched systems is presented. Then, Chapter 4 presents the second contribution, a data-driven method for construction of the Koopman Operator. Afterwards, Chapter 5 presents the Koopman Operator used in combination with Deep Learning methods and demonstrates a novel training method, the third contribution, that is used in combination with Direct Encoding of the Koopman Operator to handle unstable nonlinear systems. Concluding thoughts and future directions are offered in Chapter 6.

Chapter 2

Koopman Operator

In this chapter, we present the various methods of obtaining linear models based on the Koopman Operator framework. The basis of these methods are in lifted linearization: creating a linear system composed of a larger state space than required to represent the original nonlinear system. We begin with a brief introduction. Afterwards, the Koopman Operator framework is discussed. Following the discussed of the Koopman Operator framework, two types of methods for construction of linear systems based on the Koopman Operator are introduced. The first type mentioned is Dynamic Mode Decomposition and variations of it; these are Least Squares Estimation-based methods. The other type is Direct Encoding, which is an alternative approach for formulating a linear system based on the Koopman operator.

2.1 Introduction

The original paper regarding the Koopman Operator was written in 1931, detailing and arguing about the existence of a linear operator that transforms nonlinear systems into linear systems [23]. Several years later, Dynamic Mode Decomposition (DMD) was formulated in presented in 2010 as a method that would be applied to describing fluid flows [44]. In this work, the author introduces the method utilizing the flow field at successive time steps to predict the next state at the following time step, incorporating Singular Value Decomposition (SVD) to handle modes that were

not useful for prediction. Extended Dynamic Mode Decomposition (EDMD) introduced the concept of observable functions which were nonlinear functions of state variables [50]. Simultaneously, the author referenced the Koopman Operator as a theoretical underpinning for the method, causing DMD and its variants to be synonymous with the Koopman Operator framework. Another method introduced the viability of using DMD for control [43]. After introducing the capability of using linear Koopman models as predictors for model predictive control (MPC), numerous studies were done utilizing the methodology to real systems [26]. This significant body of research, an accumulation of fifteen years of work, has been reviewed and summarized [45]. Recently, another method for formulating the Koopman Operator was produced, termed Direct Encoding (DE) [5]. This formulation significantly differs from the DMD based formulations, and is therefore detailed separately.

Before going into the mathematics involved in the formulations, we would like to offer an intuitive explanation of the Koopman Operator framework and the methods that use the framework as an underpinning. The Koopman Operator framework can be thought of as a generalization of a Fourier transform. In a Fourier transform, a nonlinear signal is decomposed into a superposition of sinusoidal functions of various frequencies and phases. In the Koopman Operator, the functions that make up the decomposition are allowed to be more general nonlinear functions satisfying the criteria of existing in a Hilbert space; this criteria is less restrictive than only allowing sine and cosine functions. Several methods exist to compute a Fourier transform of a signal, such as Fast Fourier Transform (FFT) and Discrete Fourier Transform (DFT). Similarly for the Koopman Operator, DMD, its variants, and DE are all methods of obtaining Koopman Operator representations of a nonlinear system.

2.2 Koopman Operator Framework

Methods based on the Koopman Operator linearize nonlinear dynamic systems through lifting the order of the system. All formulations begin with a nonlinear dynamic equa-

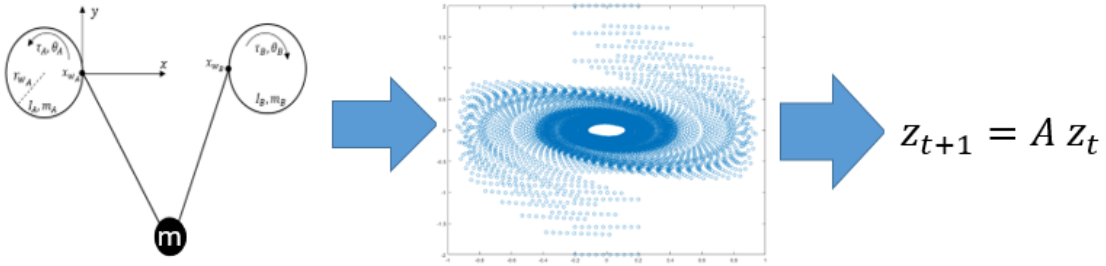

Figure 2-1: Flowchart of formulating a data-driven Koopman Operator model for a system. Beginning with a nonlinear system, a set of data is obtained. With this set of data, the linear model is formed.

tion

$$x_{t+1} = f(x_t) \tag{2.1}$$

where $x \in X \subset \mathbb{R}^n$ is the independent state variable vector representing the dynamic state of the system, f is a self-map, nonlinear function $f : X \to X$, and t is the current time step.

To lift the system, observable functions, also referred to as observables, are selected. Each observable function g is a nonlinear function of the state variables. These observables exist in a Hilbert Space $\mathcal{H}$. A Hilbert space, formally, is an inner product space that is also a complete metric space with respect to the norm of the inner product. A canonical example of sets of functions that do not belong to a Hilbert space, due to not forming a complete metric space, are Cauchy sequences. Cauchy sequences converge to continuous functions, but do not satisfy the requirements of a complete metric space. In addition to this requirement that the observable functions exist in a Hilbert Space, the composition of observable functions with the nonlinear function, that is $g \circ f$ must also exist in a Hilbert Space. An example of a nonlinear dynamic system that does not satisfy this requirement is one with discontinuities, such as the hybrid system formulation for passive walkers. These requirements and situations regarding functions like this are discussed further in depth in Chapter 3.

With these observable functions, a lifted state vector, z, is created

$$z_t = \begin{bmatrix} g_1(x_t) \\ g_2(x_t) \\ \vdots \end{bmatrix} \tag{2.2}$$

where each g_i is a single observable function. In the theoretical formulation, z may be infinite dimensional. In practical applications, a truncated form of z is found based on the selection of observable functions.

With this lifted state vector, a linear model is constructed of the form

$$z_{t+1} = Az_t \tag{2.3}$$

where A is a linear transition matrix. In the infinite dimensional case, this A matrix is the Koopman Operator. The methods to construct this linear model are discussed in the following sections, beginning with formulations based on Least Squares.

2.3 Least Squares Formulations

Underpinned by the Koopman Operator theory, Dynamic Mode Decomposition (DMD) assumes the existence of a linear state transition matrix A relating z_{t+1} to z_t, and determine A by solving a least squares regression that minimizes

$$A^O = \min_A \sum_t ||z_{t+1} - Az_t||^2 \tag{2.4}$$

Singular Value Decomposition (SVD), a factorization method of matrices, is used for the least squares optimization to compute A for all DMD based methods.

The original work regarding DMD solely utilized the state at prior time steps as

observables [44] That is the lifted state vector z would be

$$z_t = \begin{bmatrix} x_t \\ x_{t-1} \\ \vdots \\ x_{t-N} \end{bmatrix} \tag{2.5}$$

To compute A using SVD, one begins with snapshots of data in the following form:

$$\mathbf{X} = \begin{bmatrix} | & | & | & & | \\ x_1 & x_2 & x_3 & \dots & x_{m-1} \\ | & | & | & & | \end{bmatrix} \tag{2.6}$$

and

$$\mathbf{X'} = \begin{bmatrix} | & | & | & & | \\ x_2 & x_3 & x_4 & \dots & x_m \\ | & | & | & & | \end{bmatrix} \tag{2.7}$$

such that there are m snapshots and the above variables satisfy the equation $\mathbf{X'} = A\mathbf{X}$. $\mathbf{X'}$ is the set of state variables corresponding to X at the next time step. The decomposition using SVD results in

$$\mathbf{U\Sigma V}^* = \mathbf{X} \tag{2.8}$$

where $\mathbf{U} \in \mathbb{R}^{nxn}, \Sigma \in \mathbb{R}^{nxm-1}, V^* \in \mathbb{R}^{m-1xm-1}$ are the left singular vectors, the singular values of $\mathbf{X}$, and the right singular vectors respectively. The $*$ represents the complex conjugate transpose operation.

From these matrices, A can be computed as

$$A = \mathbf{X'V\Sigma^{-1}U}^* \tag{2.9}$$

To calculate the dynamic modes of the resulting A matrix, the eigendecomposition is taken. That is, the eigenvalues λ and eigenvectors v are calculated such that they

satisfy

$$Av = \lambda v \tag{2.10}$$

The dynamic modes of A for an individual DMD eigenvalue λ are given by

$$\psi = Uv \tag{2.11}$$

where ψ is the dynamic mode.

At the time of writing this thesis, Extended Dynamic Mode Decomposition (EDMD) is commonly accepted method for construction of a lifted linear model of a nonlinear system. In EDMD, observables are experimentally obtained or simulated from the governing equation of the system. These observables are then augmented with real-valued nonlinear functions of the independent state vector x_t [50]. Collectively, a high-dimensional state vector z_t is formed:

$$z_t = \begin{bmatrix} g_1(x_t) \\ g_2(x_t) \\ \vdots \\ g_m(x_t) \end{bmatrix} \tag{2.12}$$

where m is the order of the lifted state corresponding to the number of observable functions. In EDMD, the calculation of the linear state transition matrix A is still found using SVD.

This formulation was extended to include applications to systems with exogenous inputs as well [43]. In that formulation, the cost function that is minimized is instead

$$\min \sum ||z(x_{t+1}) - (Az(x_t) + Bu_t)||^2 \tag{2.13}$$

where u_t is a vector of exogenous inputs. The solution in this case equates to

$$\begin{bmatrix} A & B \end{bmatrix} = FG^{\dagger} \tag{2.14}$$

where

$$F = [z_2, ..., z_{k+1}] \tag{2.15}$$

$$G = \begin{bmatrix} z_1, ..., z_k \\ u_1, ..., u_k \end{bmatrix} \tag{2.16}$$

and $G^\dagger$ is the pseudoinverse of the G matrix. In this formulation, the pseudoinverse can still be calculated using SVD. In general, the usage of SVD is done with truncation of the total order of the system in mind as that leads to performance improvements in some cases.

Both DMD and EDMD are data-driven methods. In the next section, we will introduce an analytical method, known as Direct Encoding.

2.4 Direct Encoding from Nonlinear Mapping

An alternative to the least squares estimate and EDMD is to obtain the exact A matrix by directly encoding the self-map, nonlinear state transition function $f(x)$ with an independent and complete set of observable functions through inner product computations. This Direct Encoding method is introduced next, while the full proof can be found in [5]. Notably, a set of assumptions are presented in the following formulation. An instance where these assumptions, such as Hilbert space or continuity assumptions of functions, do not hold is explored further in Section 3.1.1.

Let $[\phi_1, \phi_2, \phi_3, ...]$ be orthonormal basis functions spanning a Hilbert space $\mathcal{H}$. Any observable g_i in the Hilbert space, $g_i \in \mathcal{H}$, can be expanded in the orthonormal basis as:

$$g_i = \sum_{j=1}^{\infty} \langle g_i, \phi_j \rangle \phi_j \tag{2.17}$$

Assuming that the self-map nonlinear function $f(x)$ is continuous, we take the composition of g_i and ϕ_j with $f(x)$ on both sides of eq. (2.17).

$$g_i \circ f = \sum_{j=1}^{\infty} \langle g_i, \phi_j \rangle (\phi_j \circ f) \tag{2.18}$$

We further assume that

$$\phi_j \circ f \in \mathcal{H} \quad \forall j \tag{2.19}$$

and $[g_1, g_2, g_3, ...]$ also form a set of orthonormal basis functions spanning the Hilbert space. Then,

$$\phi_j \circ f = \sum_{k=1}^{\infty} \langle \phi_j \circ f, g_k \rangle g_k \tag{2.20}$$

Substituting eq. (2.20) to eq. (2.18) yields

$$g_i \circ f = \sum_{j=1}^{\infty} \langle g_i, \phi_j \rangle \sum_{k=1}^{\infty} \langle \phi_j \circ f, g_k \rangle g_k \tag{2.21}$$

Concatenating $g_1, g_2, g_3, ...$ and $g_1 \circ f, g_2 \circ f, g_3 \circ f ...$ in infinite dimensional column vectors, respectively,

$$\bar{z}_t = \begin{bmatrix} g_1(x_t) \\ g_2(x_t) \\ \vdots \end{bmatrix}, \bar{z}_{t+1} = \begin{bmatrix} g_1 \circ f(x_t) \\ g_2 \circ f(x_t) \\ \vdots \end{bmatrix} \tag{2.22}$$

Eq. (2.21) can be written in matrix and vector form.

$$\bar{z}_{t+1} = \bar{A} \bar{z}_t \tag{2.23}$$

where $\bar{A}$ is an infinite dimensional matrix consisting of the inner products involved in eq. (2.21),

$$\bar{A} = \sum_{j=1}^{\infty} \begin{bmatrix} \langle g_1, \phi_j \rangle \\ \langle g_2, \phi_j \rangle \\ \vdots \end{bmatrix} \begin{bmatrix} \langle \phi_j \circ f, g_1 \rangle & \langle \phi_j \circ f, g_2 \rangle & \dots \end{bmatrix} \tag{2.24}$$

Eq. (2.23) manifests that the state lifted to the infinite dimensional space $\bar{z}_t$ makes linear state transition with matrix $\bar{A}$.

The observables $g_1, g_2, g_3, ...$ were assumed to be orthonormal basis functions in the above derivation. This assumption can be relaxed to an independent and complete

set of basis functions spanning the Hilbert space. Hereafter, let $[g_1, g_2, g_3, ...]$ be an independent and complete set of basis functions spanning the Hilbert space. Using the orthonormal basis functions $[\phi_1, \phi_2, \phi_3, ...]$, each observable can be expanded to $g_i = \sum_{j=1}^{\infty} \langle g_i, \phi_j \rangle \phi_j$. This implies that there is a linear relationship between $g_i's$ and $\phi_j's$.

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \end{bmatrix} = C \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \end{bmatrix} \tag{2.25}$$

where

$$C = \begin{bmatrix} \langle g_1, \phi_1 \rangle & \langle g_1, \phi_2 \rangle & \dots \\ \langle g_2, \phi_1 \rangle & \langle g_2, \phi_2 \rangle & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{2.26}$$

Note that, as long as $[g_1, g_2, g_3, ...]$ span the Hilbert space, the above matrix C is non-singular and therefore invertible. This allows us to extend Eq. (2.23) to one in terms of the independent and complete set of observables.

$$z_{t+1} = A_f z_t \tag{2.27}$$

where the two matrices are related as

$$A_f = C \bar{A} C^{-1} \tag{2.28}$$

Notably, once the existence of the linear state transition is guaranteed in eq. (2.27), the matrix A_f can be obtained without use of the orthonormal basis functions and the matrix C and its inverse. It can be computed directly from the self-map, state transition function $f(x)$ and an independent and complete set of observables $[g_1, g_2, g_3, ...]$ through inner product computations. Post-multiplying the transpose of z_t to both sides of eq. (2.27) and integrating them over X yield:

$$\int_X z(f(x)) z^T(x) dx = A_f \int_X z(x) z^T(x) dx \tag{2.29}$$

which can be written as

$$Q = A_f R \tag{2.30}$$

where

$$Q = \begin{bmatrix} \langle g_1 \circ f, g_1 \rangle & \langle g_1 \circ f, g_2 \rangle & \dots \\ \langle g_2 \circ f, g_1 \rangle & \langle g_2 \circ f, g_2 \rangle & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{2.31}$$

$$R = \begin{bmatrix} \langle g_1, g_1 \rangle & \langle g_1, g_2 \rangle & \dots \\ \langle g_2, g_1 \rangle & \langle g_2, g_2 \rangle & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{2.32}$$

$$\tag{2.33}$$

Because the observables are independent, the matrix R is non-singular. Therefore, the matrix A_f is given by

$$A_f = QR^{-1} \tag{2.34}$$

This formula for obtaining the matrix A_f directly from the governing nonlinear state equation with the function $f(x)$ and the independent observables through inner products, which are guaranteed to exist in Hilbert space $\mathcal{H}$, is the Direct Encoding method.

In practical applications, a finite number of observable functions g are selected. After selecting these observable functions, the elements of Q and R can be calculated as each of the elements are inner products. For example, an element of Q can be calculated as

$$Q_{ij} = \langle g_i \circ f, g_j \rangle = \int_X g_i \circ f(\xi) \bar{g}_j(\xi) d\xi \tag{2.35}$$

where the $\bar{g}$ is the complex conjugate of g. Given that real valued observables are what are normally selected, this can be ignored but has been included for correctness. This calculation requires analytically calculating this integral over the domain of the state space.

Below is an example of calculating a single diagonal element of Q and R where

the nonlinear system is

$$f(x) = \sqrt{(}x) \tag{2.36}$$

and the observable function g is

$$g(x) = e^{-x^2} \tag{2.37}$$

In this case, the corresponding diagonal element of R for this observable would be

$$\langle g, g \rangle = \int_{-\infty}^{\infty} g(\xi)^2 d\xi = \int_{-\infty}^{\infty} (e^{-\xi^2})(e^{-\xi^2}) d\xi = \sqrt{\frac{\pi}{2}} \tag{2.38}$$

and the element of Q would be

$$\langle g \circ f, g \rangle = \int_{-\infty}^{\infty} g \circ f(\xi) \bar{g}(\xi) d\xi = \int_{-\infty}^{\infty} (e^{-\xi})(e^{-\xi^2}) d\xi = \sqrt[4]{e}\sqrt{\pi} \tag{2.39}$$

Chapter 3

Modeling Hybrid Dynamic Systems with the Koopman Operator

The concern of this chapter regards applications of the Koopman Operator to **hybrid dynamic systems**, a subset of nonlinear dynamic systems that are difficult to analyze and control. Hybrid dynamic systems is a semantically overloaded term, in that it has several meanings even within one context. For the remainder of this work, our use of the term "hybrid systems" refers to nonlinear systems with segmented dynamics that may exhibit both continuous and discrete dynamics. Prior work addressing the use of the Koopman Operator with hybrid systems is also summarized here. Afterwards, the contribution is presented, where the Koopman Operator is applied to construct a linear model for the sake of Model Predictive Control (MPC), based on prior art [25]. It has been shown that MPC using the Koopman linearized model as dynamic constraints is comparable in performance and sometimes outperforms MPC using the real nonlinear system, but this was done for more benign classes of nonlinear systems [21].

The chapter will begin with an introduction into hybrid systems in robotics and in the context of the Koopman Operator. In that section, we address what nonlinear systems the Koopman Operator is applicable to, and what characteristics prohibit its use, as well as introduce the concepts (MPC and the modeling techniques used for the system of interest) that necessary to understand the experiments that follow.

The contribution articulated in this chapter is utilization of MPC to control a hybrid system. With this contribution, numerical experiments are presented. Finally, concluding thoughts are offered at the end of the chapter.

3.1 Introduction

A plethora of robotic systems can be defined as systems with segmented domains. An example of a class of robotic designs that meet this category are cable driven robots, which are known have unique characteristics. Previous designs have ranged from very simple cranes, to more complex tensegrity robots which are composed of both tension elements and compression elements [10, 30]. There are several advantages to using cable based designs: they are lightweight parallel manipulators, capable of high acceleration and velocities, and may have very large workspaces [17]. However, the cables can only pull, not push. Thus, more complex designs may increase the number of cables, making the model more difficult to analyze, but making the system more feasible to control. The dynamics of these systems is similar to that of Coloumb friction.

The prior art regarding controls in the field of cable driven robotics is vast. Some of these methods target specific dynamics of the system, such as antisway control [28]. This is done through a variety of means: linear control through a standard linearization [49], feedback linearization [24], geometric control [29], and data driven controllers [31]. These works also deal with a variety of different configurations of the system: the standard single cable configuration, multiple cable configurations, and variations of those configurations involving mobile suspension points like quadrotors. In these works, it is usually assumed that the cables are in tension. However, there are several situations neglected when not considering the dynamics of the system when the cables become slack. By including this behavior, the system becomes more complex and and difficult to model; the dynamics resemble a hybrid system due to the significant difference in speed of the dynamics. As the primary focus of this paper is control, we simulate a simple two-cable winch design that has no redundancies and

allows for the cables to all become slack while the load is in motion.

There has also been work on tracking of time variant systems using Koopman based methods through online versions of dynamic mode decomposition [2,52]. Other work involving online learning of a Koopman model has been done with active learning, and using recursive least squares to update the model [1]. In general, these works train a model based on a set of data and apply this model on the system that the data came from. With that in mind, a problem arises. One of the key benefits of using cable driven robotic systems is the ability to change the load that is carried in the case of crane robots, which causes a significant and sudden change in the dynamics of the system.

We attempt to model this segmented dynamic system behavior as a linear system using the Koopman Operator. First, we introduce the nuances of hybrid systems, and then describe criteria presented [5], regarding the conditions for existence of the Koopman Operator for systems like this. Through modeling the system linearly in a higher dimension, we are able to embed both the incredibly fast dynamics of the cables going into tension and the slow dynamics of the motion of the winches and projection motion of the mass into the lifted space and then apply MPC to the system without explicitly modeling the discrete dynamic changes that would typically make the system difficult to control [37]. This allows the controller to quickly determine the control sequence of the system, and approximate when the discrete dynamics should take place according to the optimization problem. We demonstrate this method on a simulated system with applications to cable suspended systems.

3.1.1 Hybrid Dynamic Systems and the Koopman Operator

In this work, we define Hybrid Dynamic Systems as systems that exhibit both discrete and continuous dynamics. Specifically, the continuous dynamics can be defined as

$$\dot{x} = \begin{cases} f_1(x), x \in C_1 \\ f_2(x), x \in C_2 \end{cases} \tag{3.1}$$

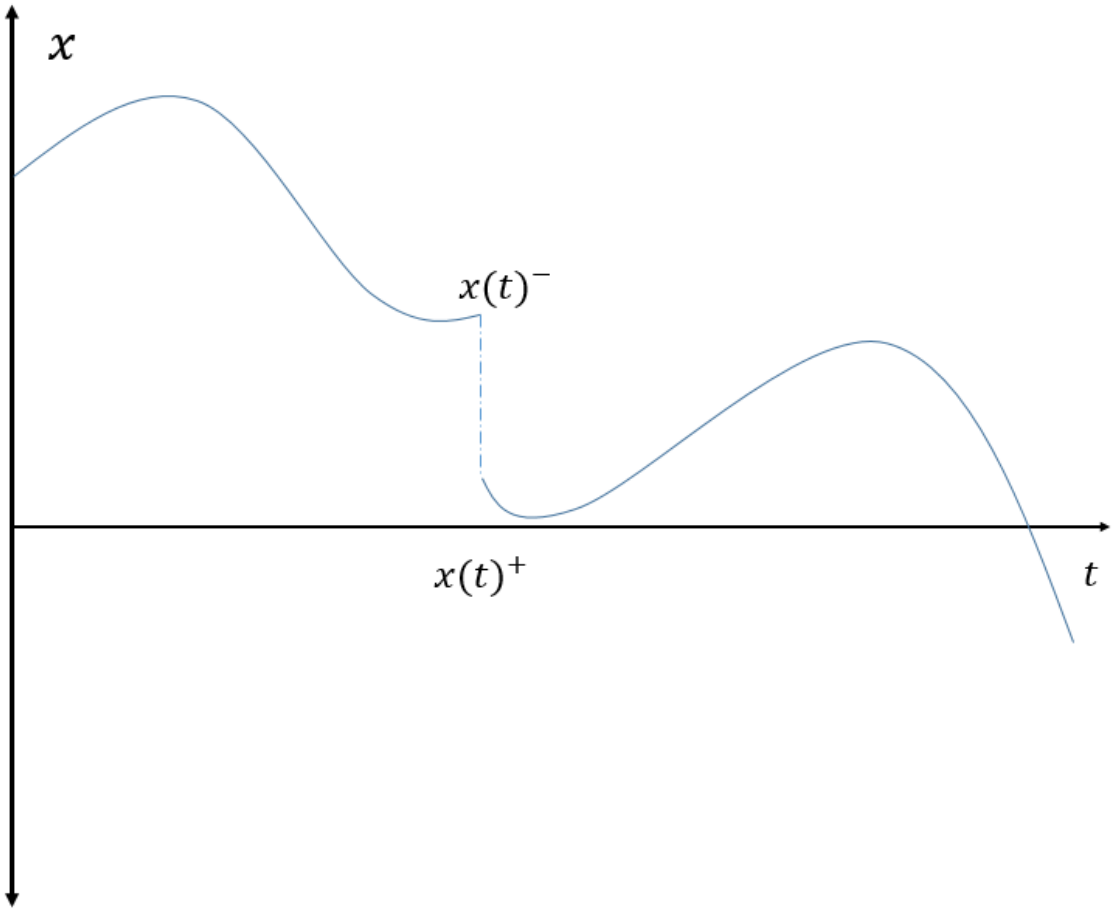

Figure 3-1: Example of a trajectory for a hybrid system that undergoes a discrete jump.

where f_1 and f_2 are continuous functions, and C_1 and C_2 are the domains of the respective dynamics.

The discrete dynamics are defined in two parts. The first part describes the the discrete jump as a function taking place over a single time step. That is

$$x^+ = g(x^-) \tag{3.2}$$

where the function g is known as the reset. The reset function is triggered when a specific function, known as a guard function $\phi(x)$, equals 0. The trajectory of such a dynamic system is visualized in Fig. 3-1.

An example of a physical system that is typically modeled as a hybrid system is

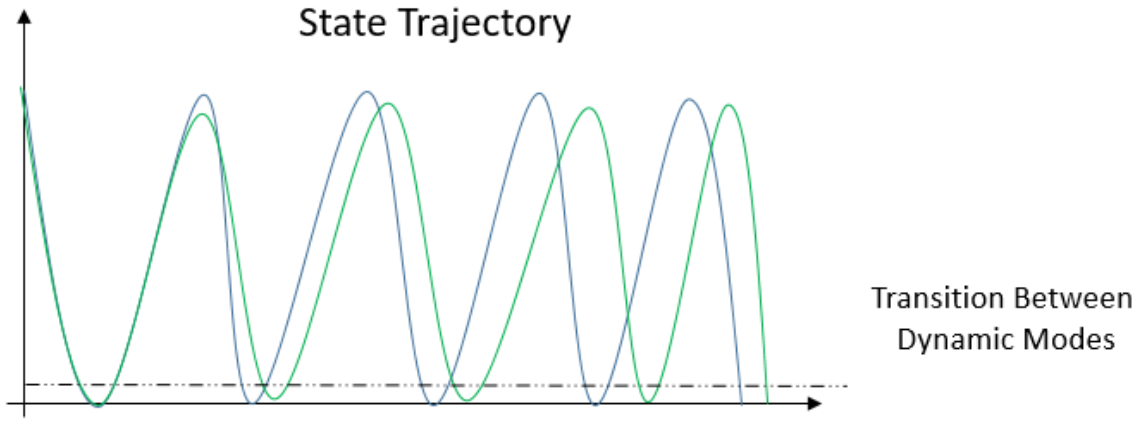

Figure 3-2: Example of how changing the timing of transitions between different dynamic domains changes a controlled trajectory.

the canonical passive walker system, illustrated in Fig. 3-3.

For the passive walker, it experiences continuous dynamics between steps:

$$\ddot{\theta} - \sin\theta - \gamma = 0 \tag{3.3}$$

$$\ddot{\theta} - \ddot{\psi} + \dot{\theta}^2 \sin\psi - \cos(\theta - \gamma)\sin\psi = 0 \tag{3.4}$$

For discrete dynamics, it experiences a discrete reset map of:

$$\begin{bmatrix} \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix}^+ = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & \cos 2\theta & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & cos2\theta(1 - cos2\theta) & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix}^- \tag{3.5}$$

with a guard function of:

$$\phi(x) = \psi - 2\theta \tag{3.6}$$

It is crucial to note that the analysis and control of such systems is incredibly difficult due to the inclusion of these discrete jumps. The analysis is problematic for similar reasons to analyzing any nonlinear system; the analysis hinges on finding a Lyapunov function that is capable of representing the system which is inherently difficult as the requirements of such a function are not difficult to satisfy. In the

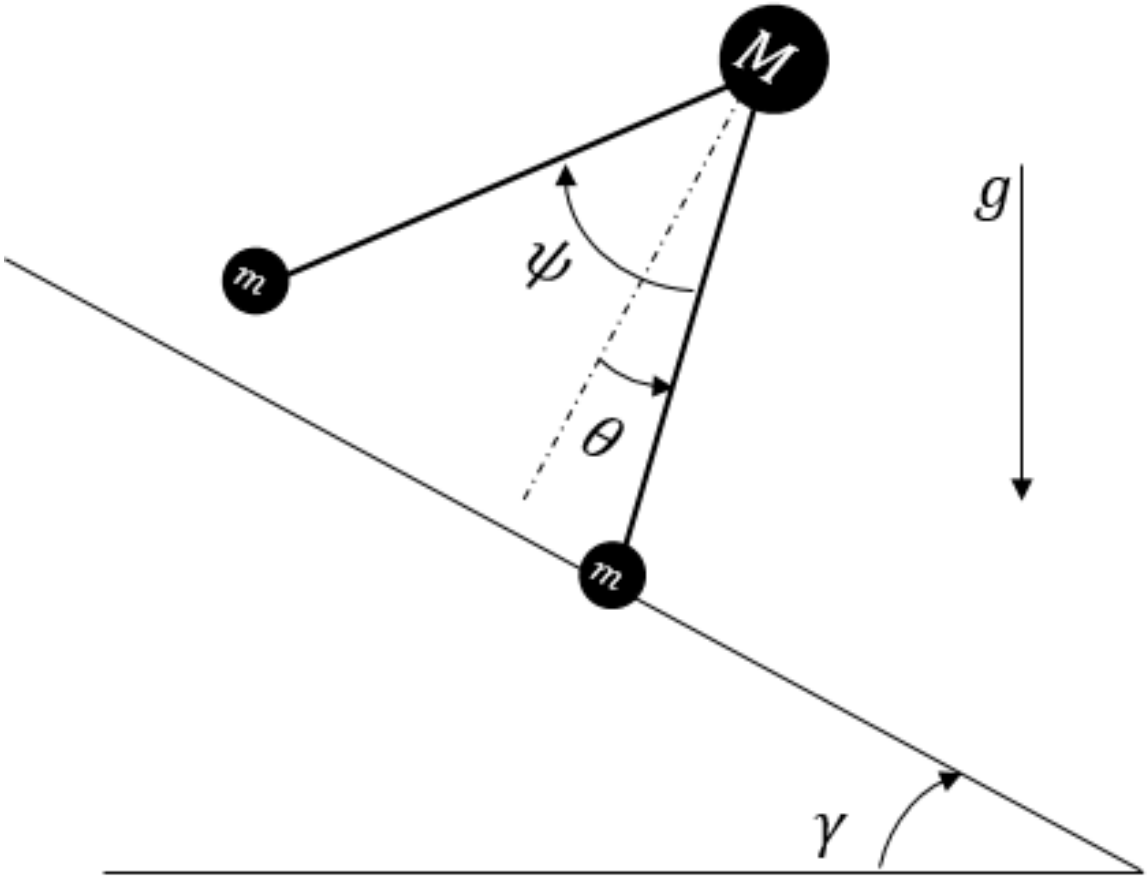

Figure 3-3: Illustration of the simplest passive walker system.

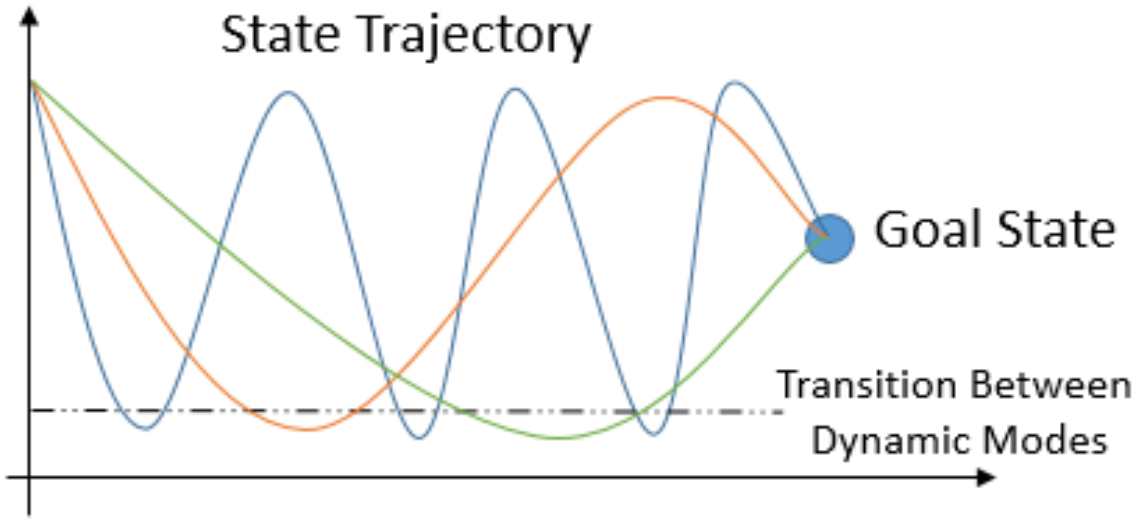

Figure 3-4: Example of how changing the number of allowed transitions between different dynamic domains changes a controlled trajectory. The dotted line indicates the boundary between the dynamic domains.

case of control, two main issues arise even in the case where the reset function is negligible, as in the case of switched systems. The first issue is one of number of switches, or transitions between types of dynamics. As illustrated in Fig. 3-4, by changing the number of different dynamic transitions allowed over a trajectory, the optimal trajectory can drastically change. Similarly, by varying the timing of these transitions, a domino effect can occur causing drastic differences in trajectory, shown in Fig. 3-2.

Prior work that attempts to control these systems introduce frameworks for model predictive control [3,9]. These frameworks address the number of transitions problem and timing problems by restricting the total number and the range of possible timings that the optimizer solves for. However, these workarounds that are proposed do not alleviate the underlying issue that is solving the optimal control problem for hybrid systems.

As such, linearizing such systems with the application of the Koopman Operator would be desired, as linear systems are far easier to analyze. In [5], weak existence conditions are provided for finite-dimensional approximations of the Koopman Operator, allowing for the application to hybrid systems. In the formulation of the

Koopman Operator, a strong condition is required, $\phi_i \circ F \in \mathcal{H}, \forall i \in \mathcal{N}$, where $\mathcal{N}$ is the set of all integers greater than 1. By replacing this condition with a weaker condition, $\phi_i \circ F \in \mathcal{H}, i < \infty$, we are enabled in producing a finite-dimensional approximation of the operator for some arbitrary, finite i. This yields a rectangular linear state transition matrix, dubbed $A_{m\infty}$ that satisfies

$$\begin{bmatrix} \phi_1 \circ F \\ \phi_2 \circ F \\ \vdots \\ \phi_m \circ F \end{bmatrix} = A_{m\infty} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \vdots \end{bmatrix} \tag{3.7}$$

where

$$A_{m\infty} = \begin{bmatrix} \langle \phi_1 \circ F, \phi_1 \rangle & \langle \phi_1 \circ F, \phi_2 \rangle & \dots \\ \langle \phi_2 \circ F, \phi_1 \rangle & \langle \phi_2 \circ F, \phi_2 \rangle & \dots \\ \vdots & \vdots & \dots \\ \langle \phi_m \circ F, \phi_1 \rangle & \langle \phi_m \circ F, \phi_2 \rangle & \dots \end{bmatrix} \tag{3.8}$$

However, this method is only required for hybrid systems that contain these discrete jumps. For a continuous system with multiple domains, termed a switched system, the existence of the Koopman Operator is not problematic. That type of system, specifically a switched system with a control input, is the subject of the rest of this chapter.

3.1.2 Model Predictive Control

The cable suspension system is a complex system as shown in its original dynamical model. Once a cable goes slack, the mass freely drops, followed by an impact at the instant when the cable becomes taut. It tends to bounce back, unless the velocity in the direction of the cable is zero. It should be noted that, once the cable goes slack, the winch of the cable is essentially disconnected and becomes unable to influence the motion of the mass. In this sense, the input to the winch loses the ability to influence the all states of the system. To maintain this ability, the robot controller

must reduce the impact and extend the controllable duration by keeping the cable taut. To find such an intelligent, skillful control action, the robot must be able to predict the dynamic behavior, in particular, the consequence of impact and bouncing. Here, we consider Model Predictive Control (MPC) for realizing such skillful actions. With MPC we can expect that the robot can find an optimal control sequence, which would minimize a potential impact and retain the ability to manipulate the mass.

For the model predictive control section, we solve an optimization problem of the form

$$\min V = \phi(x_N(t)) + \sum_{i=0}^{N-1} \ell(x_i(t), u_i(t))d\tau \tag{3.9}$$

$$s.t. \qquad x_{i+1}(t) = x_i(t) + f(x_i(t), u_i(t))d\tau \tag{3.10}$$

$$x_0(t) = x_0 \tag{3.11}$$

$$C(x_i(t), u_i(t)) \leq 0 \tag{3.12}$$

where $d\tau = T/N$, T is the time horizon, and N is the number of time steps. We use an objective function of the form

$$\ell(x, u) = (x - x_d)^T Q(x - x_d) + u^T R u \tag{3.13}$$

$$\phi(x) = (x - x_d)^T P(x - x_d) \tag{3.14}$$

where ϕ is the end stage cost. The weight matrices used in the experiments are $P = 0$, Q is of the form

$$Q = \begin{bmatrix} Q_x & 0 \\ 0 & 0 \end{bmatrix} \tag{3.15}$$

where $Q_x \in \mathbb{R}^{n \times n}$ and is diagonal; the diagonal elements of Q_x are

$$Q_{x_d iag} = [6, 0, 0.3, 0.3, 1, 1, 0.5, 0.5] \tag{3.16}$$

R is a diagonal matrix with values $\{0.01, 0.01\}$.

No inequality constraints are used in our implementation of the optimization prob-

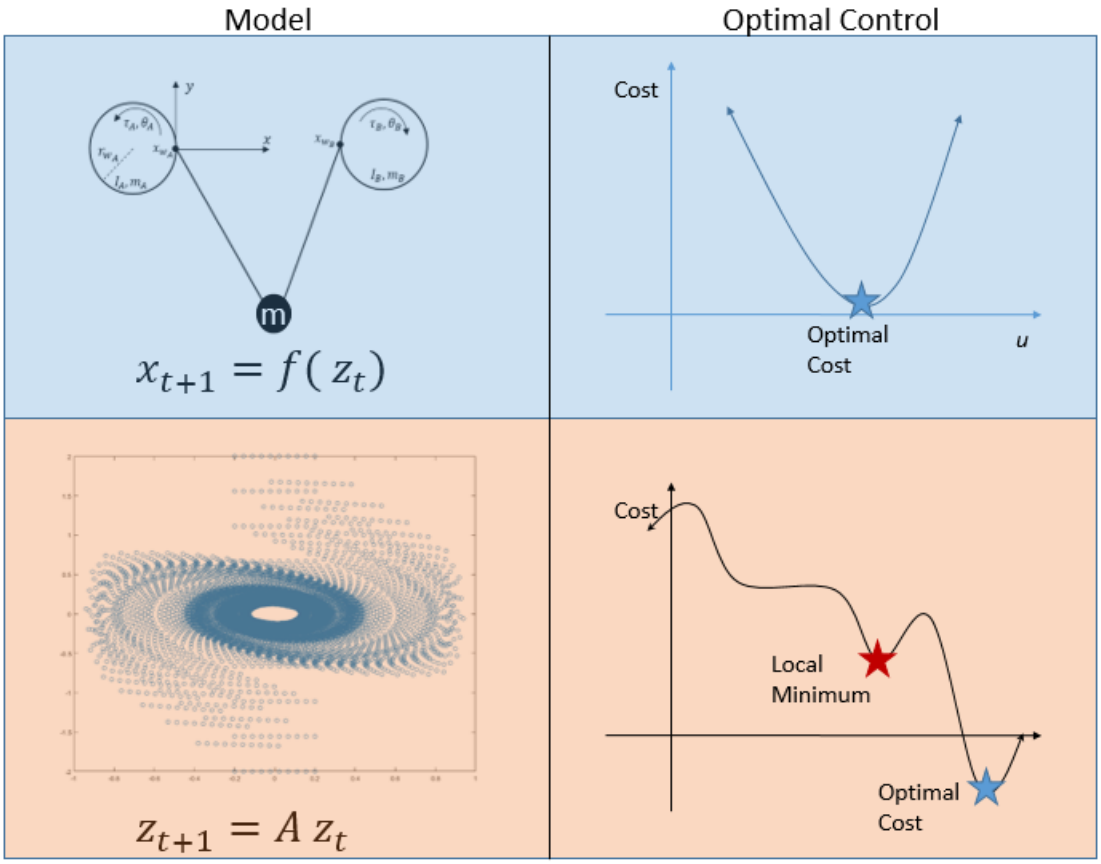

Figure 3-5: Diagram explaining the core benefits of utilizing MPC with a lifted linear model as opposed to the use of the original nonlinear model. With the lifted linear model, the optimal controller solves a simple quadratic program which finds a single optimal control input, but for a nonlinear system, there are possible local minima that are not the true optimal control input but are able to trap the optimization algorithm.

lem.

The challenge in implementing the above MPC is due to the dynamic constraints on the problem. Because the system has hybrid dynamics, it must not only optimize the control input, but also optimize the switching times between dynamic modes. Though work has been done in creating a framework to use MPC for hybrid systems [4], the framework requires denoting a set of terminal times. In essence, the problem with implementing MPC directly onto a hybrid system is that optimal sequence of modes may not be known *a priori* and leaving the solver of the optimization program to discover this sequence is computationally expensive and can sometimes

be intractable.

With this in mind, creating a linear model where the system is no longer modeled as a hybrid system with guards, modes or reset maps is very attractive. The dynamic constraints can replaced with a linear time invariant model, causing the problem to become a linear MPC problem. Linear MPC is known to be convex, and simple to solve. We no longer need to solve for a sequence of modes, nor incorporate any additional constraints on the time steps for each mode. This does require having an accurate linear model for the system throughout the time horizon of the problem, which is feasible because of the Koopman operator.

3.1.3 Modeling of Cable Suspension System

Modeling as a Switched System

The simulated system is based on a real robotic system from a prior work which utilized three winches and a very similar design [20]. The original robotic system's design is pictured in Fig. 3-6. The simulated system is a simplified version of this robotic system and a diagram for this system that can be found in Figure 3-7.

The object suspended by the cable system is treated as a point with mass m. Because the system is constrained to two dimensions, the state vector is defined as:

$$\mathbf{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & L_A & L_B & \dot{L}_A & \dot{L}_B \end{bmatrix}^T \tag{3.17}$$

where L_A and L_B are *unstretched* cable lengths and $\dot{L}_A$ and $\dot{L}_B$ are their time derivatives.

The unstretched cable lengths and their velocities are functions of the winch rotation θ_i and $\dot{\theta}_i$ respectively, where i corresponds to the cable identifier, A or B. The unstretched cable length is defined as

$$L_i = L_0 + r_{w_i}\theta_i \tag{3.18}$$

and the unstretched cable length velocity is the time derivative of the above function

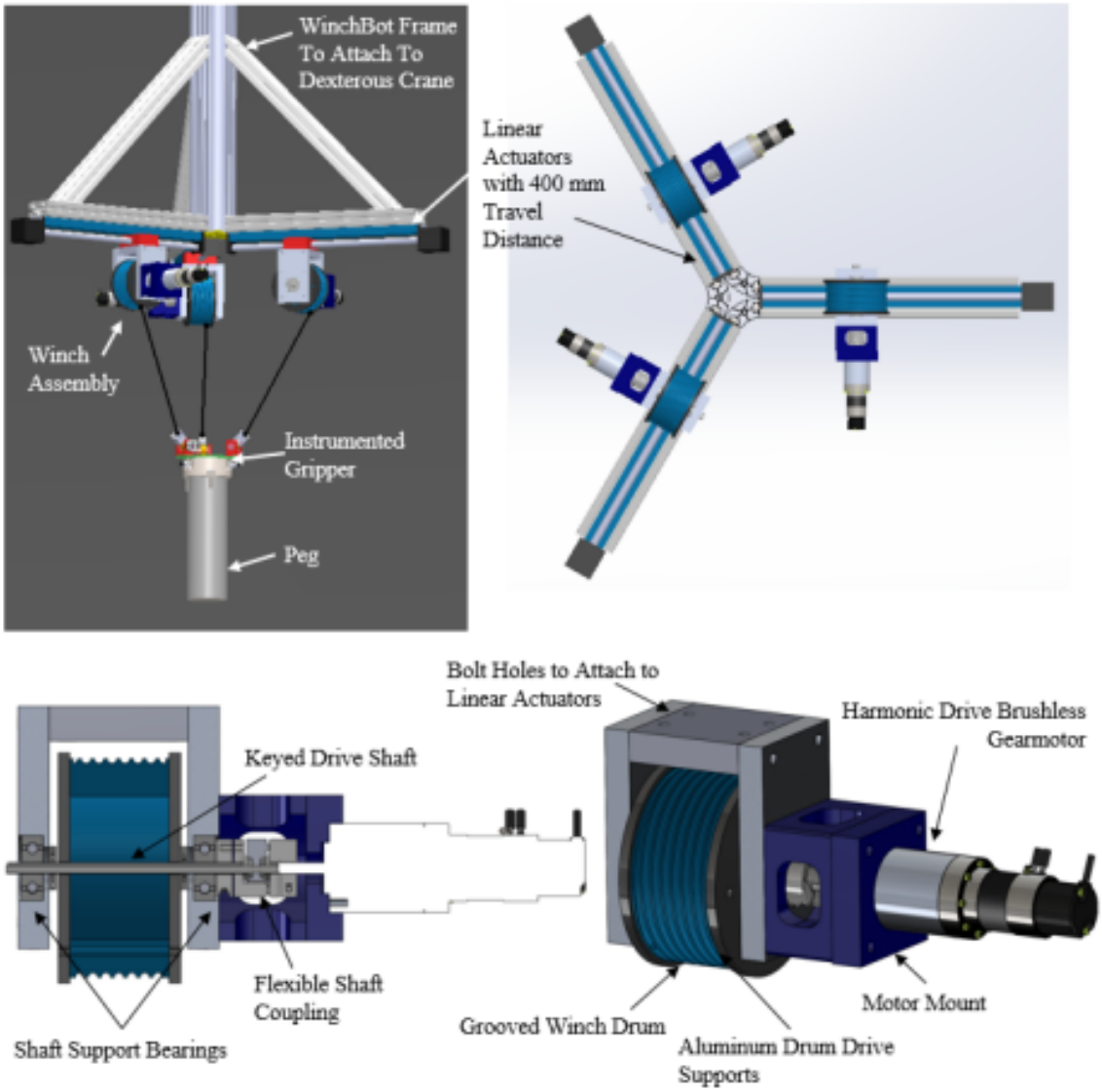

Figure 3-6: CAD diagram showing the key components of the WinchBot Winch design from [20].

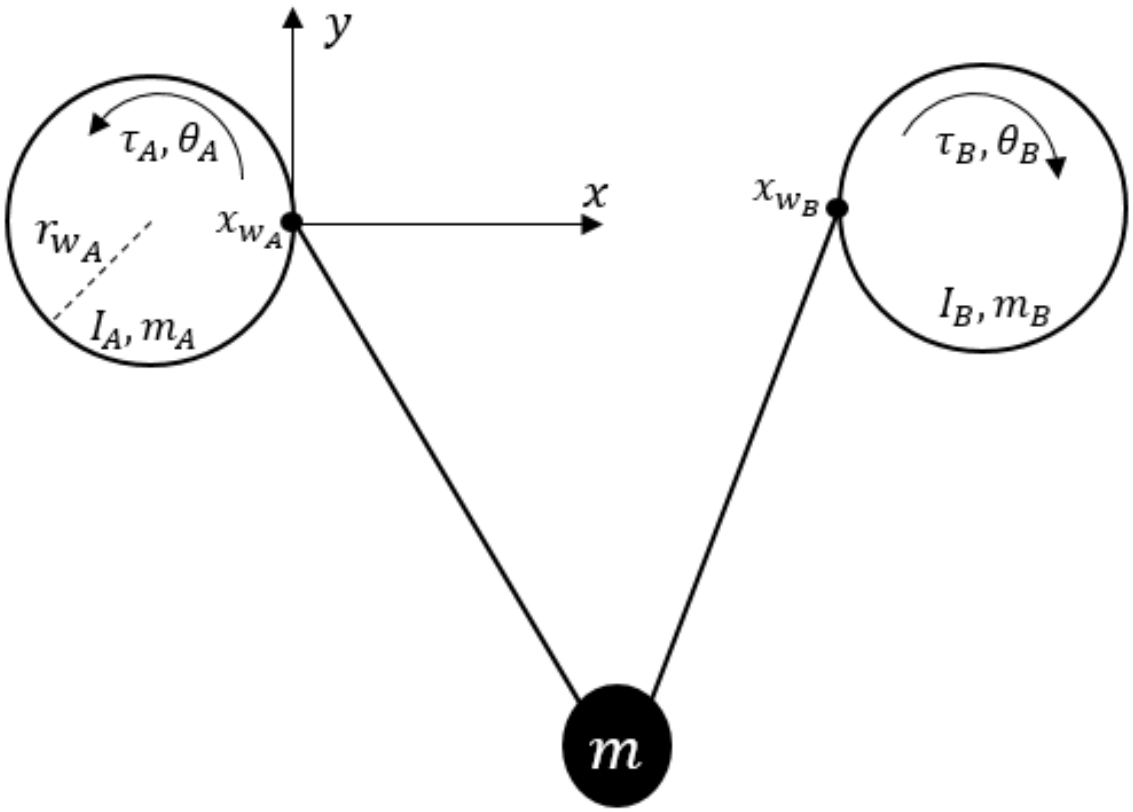

Figure 3-7: Diagram of the hybrid system involving two winches, two cables, and a suspended mass. The winches are each driven directly by a motor. The point at which the cable departs from the winch is maintained as a fixed point for simplicity. We refer to the cable on the left as cable A and the cable on the right as cable B.

and is also a state variable.

Considering this, there are four state variables associated with the winches and cables, and four state variables associated with the suspended mass. The winch positions are fixed in place, but allowed to rotate. The winch rotation dynamics are written as

$$I_i\ddot{\theta}_i = u_i - \tau_{w_i} \tag{3.19}$$

where τ_{w_i} is the torque due to the cable when in tension, and u_i is the input torque from the motor attached to the winch. This torque is defined as

$$\tau_{w_i} = r_{w_i}\mathbf{n}_{w_i} \times T_i\mathbf{n}_i \tag{3.20}$$

where r_{w_i} is the radius of the winch, and $\mathbf{n}_i$ corresponds to the unit vector in the direction of the departure point of the cable from the center of the winch.

This system's different dynamic modes can be represented visually by Fig. 3-8. By modeling the system as a hybrid system, we have two options: 1) choose to ignore the dynamics of the system when the cable suddenly goes in tension and causes the mass to "bounce", and instead model it as a discrete reset map, or 2) use the guard function to map the dynamic transition from slack to tension and tension to slack for each cable. We model the system according to the second option, in which the cables tension switches between the different dynamic domains with continuous state

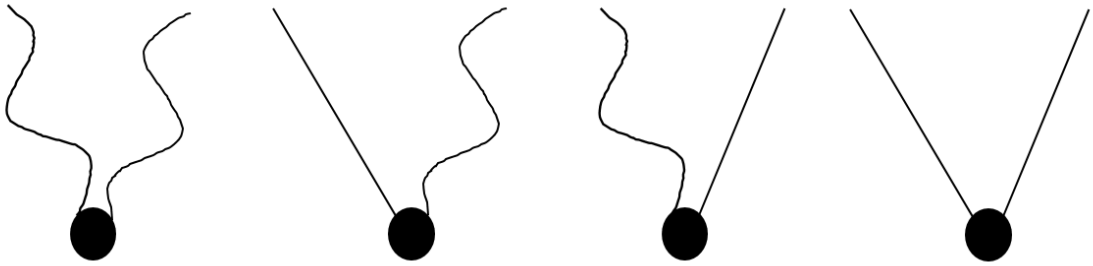

Figure 3-8: The different dynamic modes of the system that are determined by whether the cables are in tension. From left to right: both cables are in tension, cable A is in tension and cable B is slack, cable A is slack and cable B is in tension, both cables are slack.

variables. As such, we define the system's dynamic equations as

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{cases} \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{if } d_A \leq 0 \wedge d_B \leq 0 \\ T_A \mathbf{n}_A + \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{if } d_A > 0 \wedge d_B \leq 0 \\ T_B \mathbf{n}_B + \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{if } d_A \leq 0 \wedge d_B > 0 \\ T_A \mathbf{n}_A + T_B \mathbf{n}_B + \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{otherwise} \end{cases} \tag{3.21}$$

and where d_i is the elongation length of cable i subtracted by its unstretched length based on the positioning of the mass relative to the respective winch.

3.2 Cable Manipulation Based on MPC in Lifted Space

In this section the MPC formulation using lifting linearization is implemented for a multi-cable robot system. We create a realistic model of tension using experimental

data, demonstrate the accuracy of the linearized system in comparison to the full nonlinear system, and apply the lifted linear model to MPC for driving the system to specific reference states from randomized initial conditions.

The simulations are calculated with the Koopman linearized system and also compared to a DMD system which truncates the dynamic modes to a rank that contains 99% of the information based on the singular value decomposition.

3.2.1 Tension Modeling

The tension in the cables was assumed to be elastic and modeled through taking experimental data of the cables used in the real robotic system. The model does not include any damping component; this assures that no causality problem arises from lifting the space, since anti-causal observables pertain to damping elements alone [46]. This experimental data is shown in Fig. 3-9 and the parameters found from this experimental data is shown in Table 3.1.

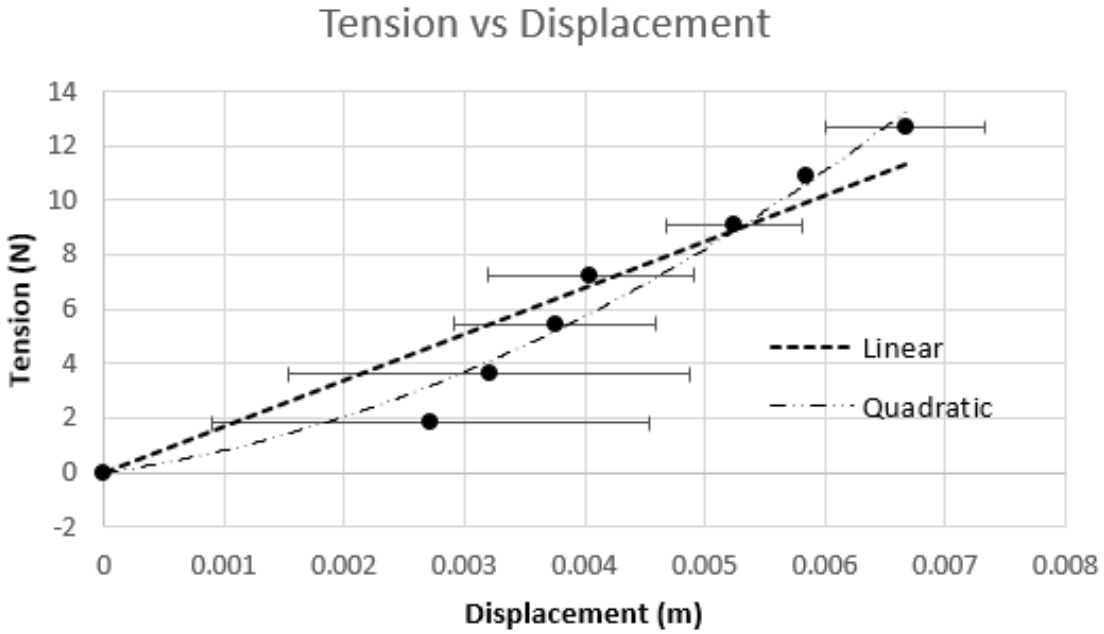

Figure 3-9: Tension-Displacement data gathered from the motivating robotic system. The data is fit to polynomial models. Models of higher than 2nd order were deemed unreasonable as they gave the possibility of negative tension. The models were also required to intersect with zero tension at zero displacement.

Table 3.1: Tension Models

Model	R^2	Equation
Linear	0.8914	$F = 1699x$
Quadratic	0.9673	$F = 206015x^2 + 613.99x$

Polynomial models were considered, and only two reasonable models were found, the linear and quadratic model. For the simulations, the linear model was chosen though it had a lower R^2 value, as it emphasized the hybrid dynamic nature of the system more than the quadratic model which was continuous and differentiable through the entire domain of the system, including “negative" displacements where the tension remains zero.

3.2.2 Model Training

Both Fig. 3-10 and 3-11 were produced for trajectories where the control law applied was a proportional controller on cable length. In Fig. 3-10, an estimation of the linearized system compared to the real system for one of the state variables is shown. In Fig. 3-11, a plot of the MSE (mean squared error) for the eight state variables is shown as a function of the length of the prediction time. As the prediction time increases, the mean squared error tends to increase. In the numerical results presented, we find that the system can be reasonably estimated using the Koopman linear system and by the DMD system, though less so, which is expected. The truncation of the system also appears to cause a significant increase in variation of the prediction error for the system, however, it is within reason for small time horizons to still be used for model predictive control.

3.2.3 Model Predictive Control

For all implementations of MPC, the time horizon was chosen to be one hundred time steps, or one second, and was selected based on the accuracy of prediction over time shown in Fig. 3-11.

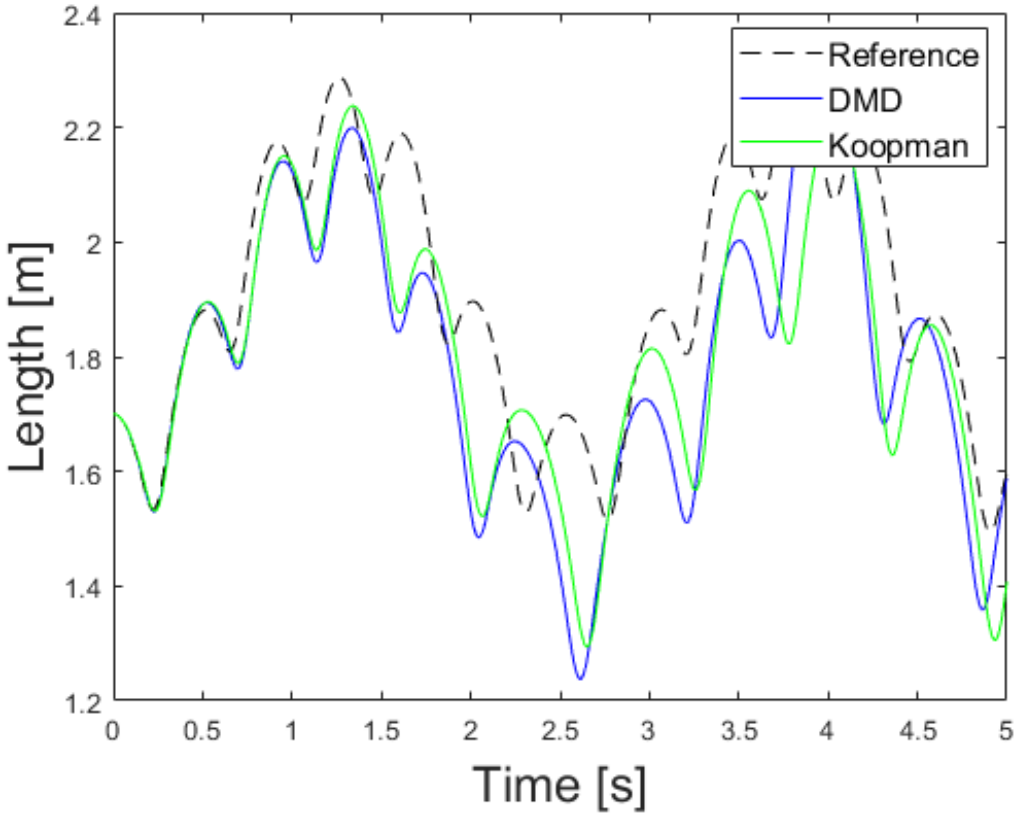

Figure 3-10: Example of the estimation provided from the Koopman linear system and the DMD system for a state variable; in this plot it is the length of cable A.

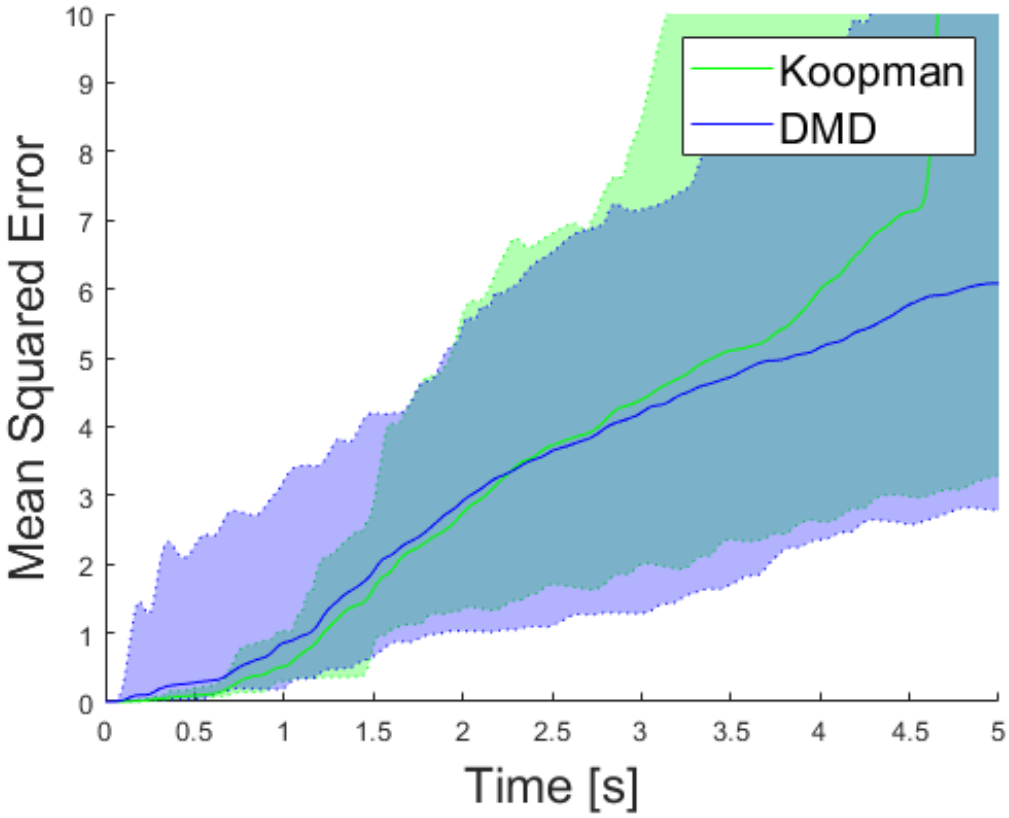

Figure 3-11: A plot of MSE over time for the linear predictor models. Each predictor begins with the correct initial state. The shaded regions represent the variation in mean squared error over twenty five different trajectories, while the solid lines represent the average mean squared error for the collection of trajectories. The plot demonstrates that for small amounts of time, the Koopman model is much more accurate than the DMD model, but after enough time has passed, both models approach the same levels of inaccuracy.

As stated previously, the weight matrices used in the experiments are $P = 0$, Q is of the form

$$Q = \begin{bmatrix} Q_x & 0 \\ 0 & 0 \end{bmatrix} \tag{3.22}$$

where $Q_x \in \mathbb{R}^{n \times n}$ and is diagonal; the diagonal elements of Q_x are

$$Q_{x_{diag}} = [6, 0, 0.3, 0.3, 1, 1, 0.5, 0.5] \tag{3.23}$$

R is a diagonal matrix with values $\{0.01, 0.01\}$. No inequality constraints are used in our implementation of the optimization problem.

In Fig. 3-12, the resultant trajectory simulated from MPC that used the Koopman linearized model as the dynamic constraints for the optimization problems is juxtaposed with the control input that was generated for cable A. The steady state error in the state variables is expected as a non-zero weight is used for the input, and the reference points require the require a significant input. Initially, the mass was placed 3 meters vertically above the goal position with both cables being slack. With MPC+Koopman, the mass trajectory smoothly converged to the goal position, as shown by the red line in the vertical position plot. For comparison, a naive PD control that controls the individual cable length without predicting the mass behavior is plotted by blue lines. Note that the mass bounced when the two cables became taut consecutively at $t = 1.1 \sim 1.7$ sec, as indicated by notches in the plots. This resulted in a pronounced bouncing motion of the mass, leading to the slow conversion of the naive cable length control. In contrast, MPC+Koopman could smoothly move the mass towards the goal. Interestingly, the MPC control input did not attempt to lengthen the cable rapidly, unlike the naive PD control, as seen in section A indicated by the orange circle. Instead, the MPC control allowed the mass to pull on the cable, and dampened the motion of of the mass. These behaviors cannot be created unless the robot can predict the nonlinear hybrid nature of the system dynamics.

As stated previously, the dynamic constraints are all that are necessary to generate a solution for this MPC problem, and no additional constraints or equations were set

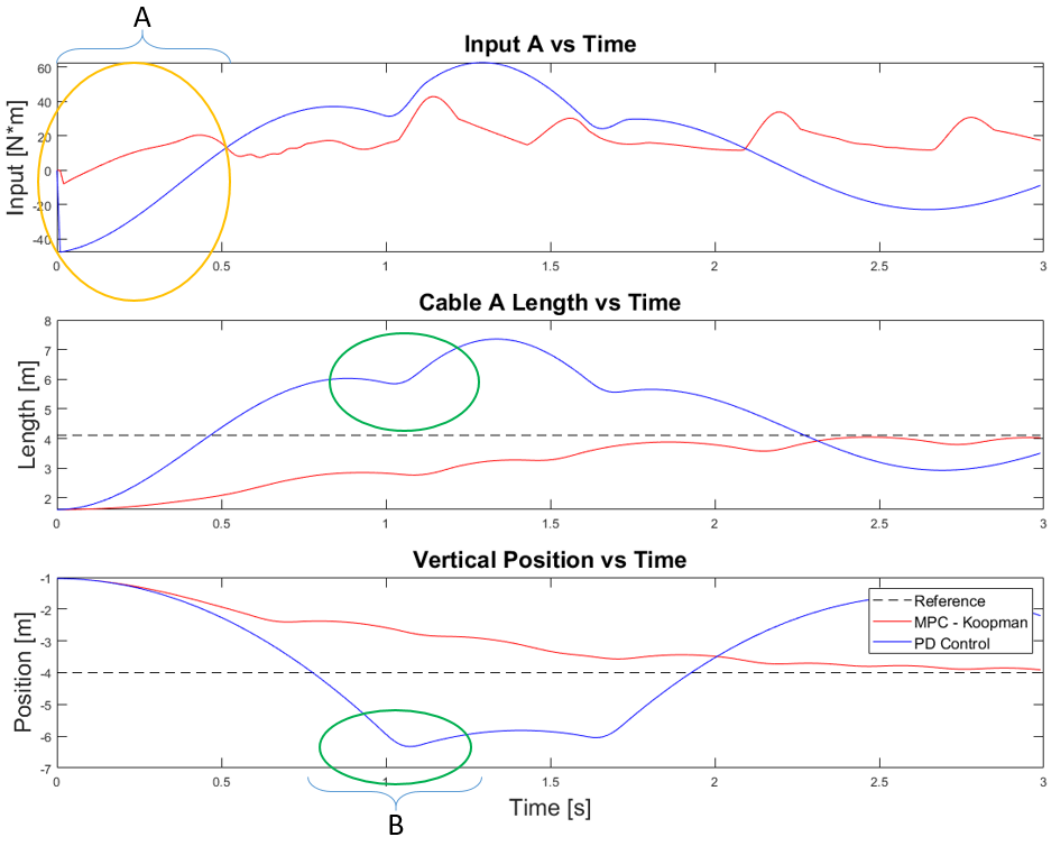

Figure 3-12: A juxtaposition of the input used for cable A and the trajectory of the length of cable A over time when using model predictive control with the Koopman linearization as the dynamic constraints. This is compared to the trajectory found from using a PD controller. In section A, denoted by the orange circle, the MPC control input does not attempt to lengthen the cable significantly like the PD controller, and instead allows the mass to pull on the cable, and dampens the motion of the mass instead. In section B, denoted by the green circle, we observe the mass disturbs the cable dynamics for the PD controller, which it is unable to reject properly.

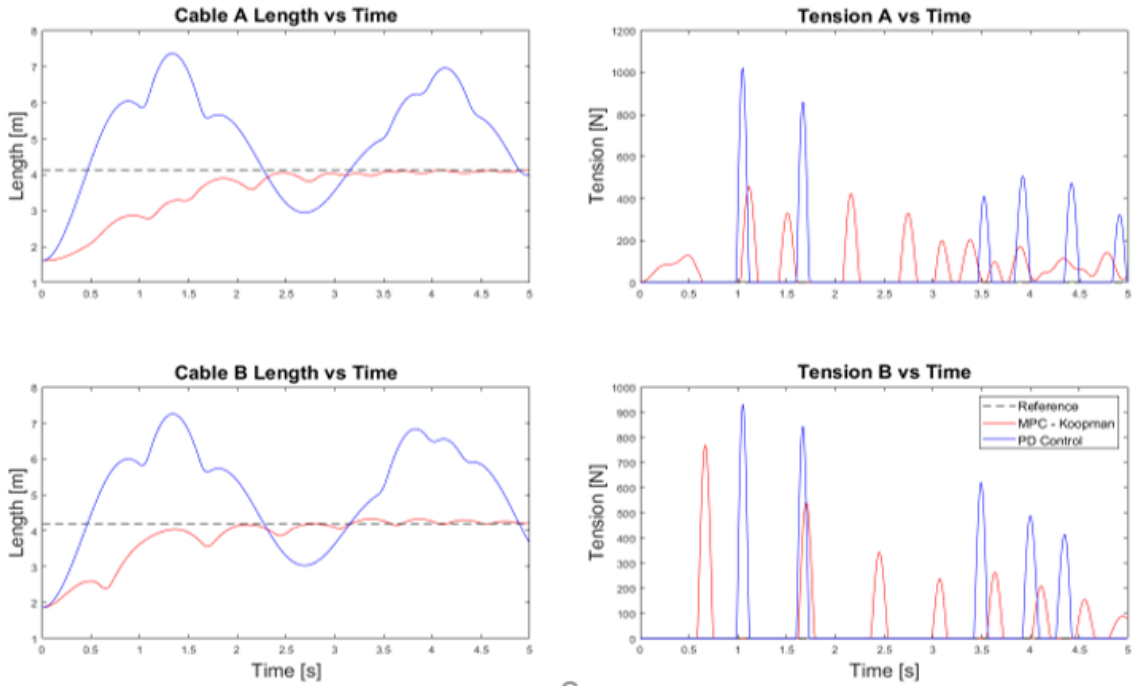

Figure 3-13: A juxtaposition of the tensions for the cables and the trajectories of the length of each cable over time when using model predictive control with the Koopman linearization as the dynamic constraints. This is compared to the trajectory found from using a PD controller.

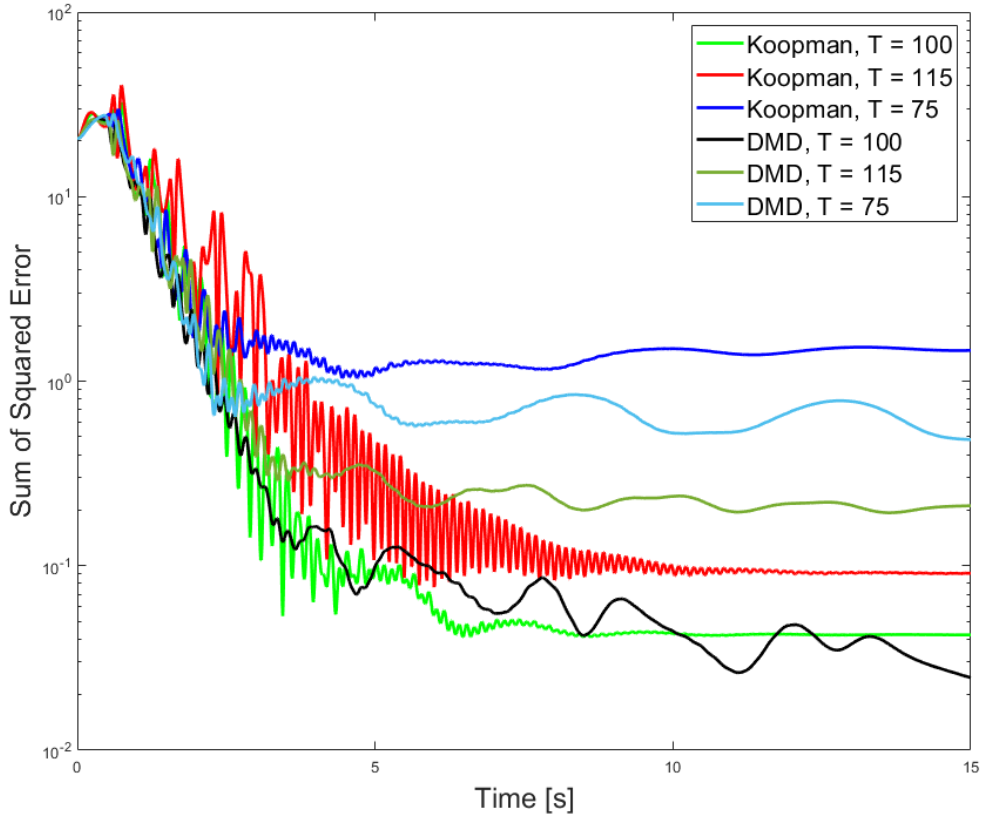

Figure 3-14: SSE (Sum of Squared Error) over the trajectory comparing an MPC scheme using the DMD model and the Koopman model as dynamic constraints. The number indicated in the legend denotes the prediction time horizon in time steps for the model. It was found that after approximately 100 time steps that MPC began performing poorly, likely due to inaccuracy of the linear model.

on time. The MPC result is successful in that it does rapidly decrease the SSE with respect to the reference trajectory as shown in Fig. 3-14; the slight differences between using the Koopman linear system as dynamic constraints and the DMD system as linear constraints is expected given prior work.

3.3 Concluding Thoughts

In this chapter, first addressed the issues of modeling hybrid systems with the Koopman Operator. We then presented a novel approach to using optimal control for hybrid systems using the Koopman Operator and applied it to an elastic cable sus-

pension system. This allowed us to use model predictive control without the difficulty of determining controllable hybrid sequences nor switching times as the hybrid dynamics were encoded into a linear time invariant system. The model predictive control input was also analyzed on its behavior, showing that it demonstrated proper model predictive behavior despite using a linear model to represent an inherently hybrid dynamic system.

Chapter 4

Data-Driven Encoding

The goal of this chapter is to address the inconsistencies between existing formulations of the Koopman Operator detailed in Chapter 2. The chapter begins with an introduction to the problem that arises due to the difference in nature of the formulations; DMD and its variants are data-driven methods and DE is an analytical method, a data-driven formulation is created based on DE. The motivation for this work is then discussed, where we articulate the reason that the difference between the methods is a cause for concern. We then introduce the novel data-driven formulation based on DE. This novel formulation serves as a significant contribution, the second contribution of this thesis, as it is shown to be robust to the distributions of data used in its formulation. Afterwards, a numerical experiment is presented that demonstrates the value of the novel method and confirms properties that are articulated in the formulation. Lastly, concluding thoughts are offered.

4.1 Introduction

A fundamental difficulty in constructing a proper linear model is data dependency. Least Squares Estimation (LSE), involved in all DMD based methods, often produces a significant bias due to the distribution of the dataset. To eliminate this dependency on data distributions, the current work takes an alternative approach to LSE.

This differs significantly from DE, presented in Chapter 2. This method directly

encodes the nonlinear function of state transition using observables as basis functions to obtain a Koopman linear model. Inner products of observable functions in composition with the nonlinear state transition function are used to construct the state transition matrix without use of LSE. While DE theoretically guarantees the exact linear model, it requires access to the nonlinear state equations, which are often not available in practical applications. The current work aims to fill the gap between DE and data-driven approaches.

There is not a simple conversion between the DMD methods and DE, even while utilizing the same observable functions, for many reasons. One of which is because DE utilizes the underlying nonlinear model in combination with the observable functions to generate the linear matrix. In addition, the methods are calculated in fundamentally different manners; DE utilizes integration over the state space, whereas DMD methods utilize least squares estimation based on data. However, the need for the underlying nonlinear model is a significant limitation for DE. Thus, there is a gap between DE and a method that can be utilized for real applications that are currently able to be handled with DMD as it is a data-driven method.

In this chapter, we begin by discussing the issues with the existing formulations of constructing the linear matrix of the Koopman Operator. We then outline the process of converting the analytical DE method to a data-driven method, termed Data-Driven Encoding (DDE). In addition, a proof of the convergence of this method to its analytical counterpart is presented along with the circumstances required for convergence. An analytical algorithm is given that allows for efficient use of the method. Numerical experiments are utilized on a simple second order dynamic system to illustrate the key differences between DDE and the traditionally used EDMD.

4.2 Motivation

The prevailing method for construction of the Koopman Operator, EDMD, is based on LSE and SVD. This method, however, cannot provide a consistent estimate; the result is highly dependent on the distribution of data as the Koopman Operator is

being approximated [32]. This dependency on distribution occurs because a core assumption of LSE is that the model structure is correct; when this assumption is violated, LSE is unable to create an unbiased estimator [16]. As the Koopman operator is truncated in practical use, this assumption does not hold.

Non-uniform data distributions inevitably occur in practical applications. For a nonlinear dynamical system with a stable equilibrium, for example, data collected from experiments and/or simulation of the system tend to be dense in the vicinity of the equilibrium, as all trajectories that begin within a region of attraction converge to the equilibrium. This problem is illustrated in Fig. 4-1, in which a situation where a large number of points is measured in a small dynamic region, and points away from the densely populated region are sparse, leading to alternative models for a single dynamic systems. Because LSE applies equal weighting to all data points, the model is heavily tuned to the behavior of the densely populated region. Of course, this dilemma does not arise when the observable functions are completely and correctly chosen as the Koopman Operator can be exactly determined in that case, but this is not the general case in applications of the Koopman Operator.

The Direct Encoding method described previously enables us to obtain the exact linear state transition matrix A through inner product computations. As the formulation is based on integration over the entire state space, there is no bias towards particular parts of the domain.

However, the original form of the Direct Encoding method utilizes the nonlinear state equation, i.e. the self-map $f(x)$, to compute the inner products. In practical applications, such a nonlinear function is not always available; only data are available. The objective of this section is to establish a computational algorithm to obtain the A matrix by numerically computing the inner products, $\langle g_i, g_j \rangle, \langle g_i \circ f, g_j \rangle$, from a given set of data.

The method presented consists of three operations.

- The integral of the inner products is reduced in range from the entire state space to the dynamic range encapsulated by the data.

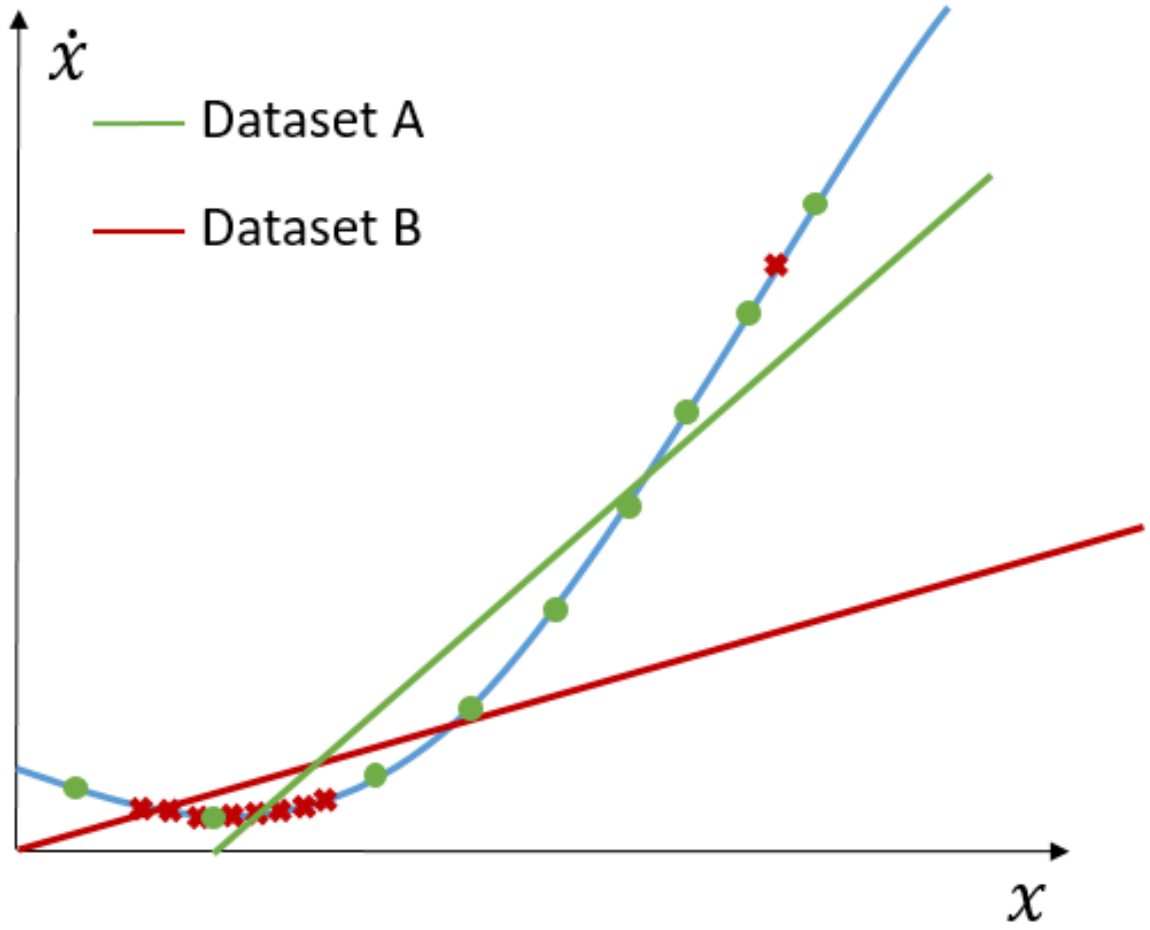

Figure 4-1: Artistic illustration of differences for a Least Squares estimation utilizing two different datasets drawn from the same underlying function. The ground truth from which data points are drawn from is in blue.

- The dynamic range is discretized with data points.
- The inner product integral is reduced to a weighted summation of the integrand evaluated at each data point multiplied by the volume Δv associated to each point.

Naturally, if data are densely populated in a small region, the discretized integral interval is small and thereby the volume also becomes small. Similarly, the volume tends to be larger where the data are sparse. In the summation, the integrand evaluated at individual data points are "weighted" by the size of the volume. This numerical inner product calculation prevents overemphasis of clustered data.

4.3 Formulation

We present the data-driven encoding method (DDE) as an alternative data-driven method to DMD for calculating a finite order approximation of the Koopman Operator. In the algorithm presented, we utilize a mesh generation algorithm, such as Delaunay Triangles [11] for an n-dimensional nonlinear system, where $n \geq 2$. Depending on the dimension of the system, different methods of mesh construction can be used interchangeably with Delaunay Triangles.

The objective of this method is to compute the matrices R and Q in (13) and (14) from data. This entails the computation of inner products:

$$\langle g_i, g_j \rangle = \int_X G_{ij}(\xi) d\xi \tag{4.1}$$

$$\langle g_i \circ f, g_j \rangle = \int_X F_{ij}(\xi) d\xi \tag{4.2}$$

where

$$G_{ij}(x) = g_i(x)\bar{g}_j(x), F_{ij}(x) = g_i[f(x)]\bar{g}_j(x) \tag{4.3}$$

are assumed to be Riemann Integrable; the functions are bounded and continuous [14].

There are two data sets used for the inner product computation. The first data set is

$$D_N = \{x_i \mid i = 1, \cdots, N; x_i \in X\} \tag{4.4}$$

Note that all the data values are finite, $|x_i| < \infty$. As such, the integral interval of the inner products is finite in computing them from the data. To define the integral interval, we consider the dynamic range of the system, X_D, determined from the data set D_N. See Fig.4-2. The dynamic range X_D is defined to be the minimum domain in the space X that includes all the data points in D_N, $X_D \supset D_N$, and that is convex. Namely, for any two states in X_D, $x, x' \in X_D$,

$$\xi = \alpha x + (1 - \alpha)x' \in X_D \tag{4.5}$$

where $0 \leq \alpha \leq 1$. Note that X_D is the smallest domain that is convex and that

includes all the data points, as shown in Fig. 4-2. Each data point x_i is mapped to

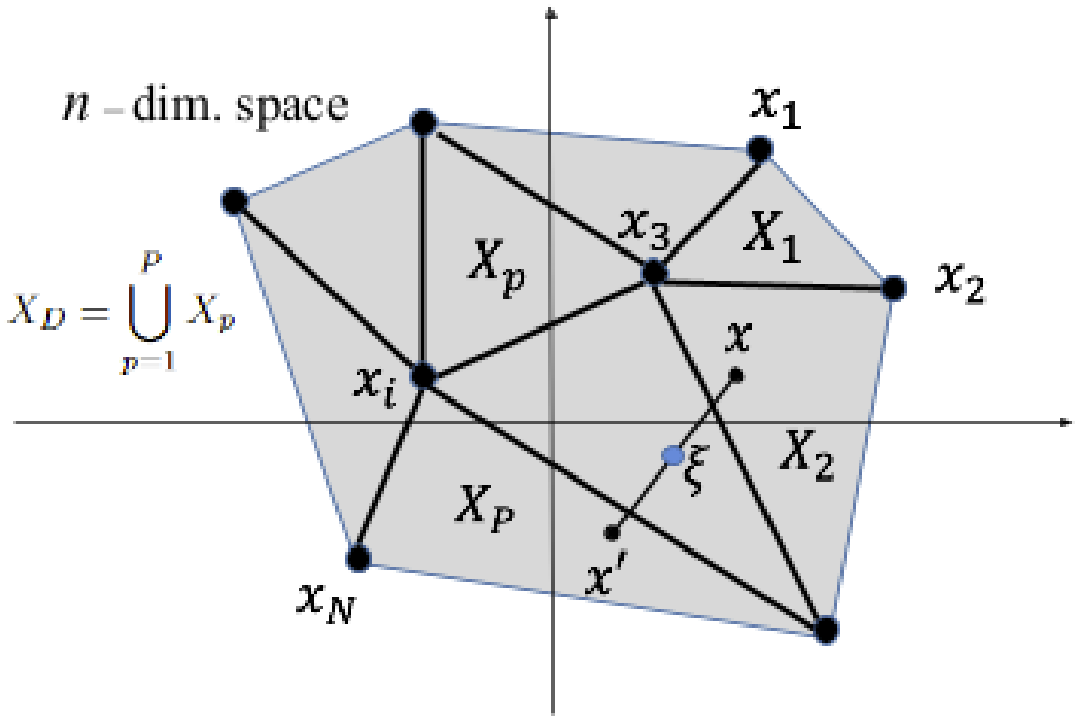

Figure 4-2: Illustration of the dynamic range of a dataset defined by the convex hull containing all points in the set, partitioned using a triangulation method. The data points are in black and the convex hull that encapsulates all data points is in grey.

$f(x_i)$, following the state transition law in eq.(2.1). We assume that the transferred state, too, stays within the same dynamic range X_D. This assumption is valid for all stable regions of a nonlinear system. In the case of an unstable system, maintaining this assumption requires discarding the data points on the perimeter of the convex hull. Collecting all the transferred states yields the second data set.

$$D_N^f = \{f(x_i) \mid i = 1, \cdots, N; x_i \in D_N\} \tag{4.6}$$

$$D_N^f \subset X_D \tag{4.7}$$

This implies that the state space of the nonlinear system under consideration is closed within the dynamic range X_D.

With this dynamic range, we redefine our objective to compute the inner products

over X_D.

$$R_{ij} = \int_{X_D} G_{ij}(x)dx \tag{4.8}$$

$$Q_{ij} = \int_{X_D} F_{ij}(x)dx \tag{4.9}$$

The integrals can be computed by partitioning the domain X_D into many segments $X_1, \cdots X_P$, as shown in Fig. 4-2.

$$X_D = \bigcup_{p=1}^{P} X_p \tag{4.10}$$

We generate these segments by applying a meshing technique to the data set D_N, where the n-dimensional coordinates of individual data points are treated as nodes of a mesh. Delaunay Triangulation, for example, generates a triangular mesh structure with desirable properties [11]. As illustrated in Fig. 4-2, each triangular element is convex and has no internal node. The volume of the dynamic range $V(X_D)$ is the sum of the volumes of all the elements.

$$V(X_D) = \sum_{p=1}^{P} \Delta v_p \tag{4.11}$$

Accordingly, the integral R_{ij} in eq.(4.8) can be segmented to

$$R_{ij} = \sum_{p=1}^{P} \int_{X_p} G_{ij}(x)dx \tag{4.12}$$

Suppose that the p-th element has K_p nodes, as shown in Fig.4-3. Renumbering these nodes 1 through K_p,

$$\{x[k_p] \Big| x[k_p] \in X_p; k_p = 1, \cdots, K_p\}, p = 1, \cdots, P \tag{4.13}$$

The integrand G_{ij} within the p-th element can be approximated to the mean of the

K_p nodes involved in the p-th element.

$$G_{ij}(x;p) \approx \bar{G}_{ij,p} = \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p]), x[k_p] \in X_p \tag{4.14}$$

If Delaunay Triangulation is used, $K_p = n + 1$. See Fig. 4-3. Substituting this into (4.8) yields the approximate value of R_{ij}.

$$\hat{R}_{ij} = \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p]) \Delta v_p \tag{4.15}$$

where

$$\Delta v_p = \int_{X_p} 1 \cdot dx \tag{4.16}$$

Similarly, each component of the matrix Q can be computed by using the same meshing.

$$\hat{Q}_{ij} = \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} F_{ij}(x[k_p]) \Delta v_p \tag{4.17}$$

Note that F_{ij} is evaluated by using the data points in both D_N^f and D_N,

$$F_{ij}(x[k_p]) = g_i[f(x[k_p])] \bar{g}_j(x[k_p]) \tag{4.18}$$

where $f(x[k_p]) \in D_N^f$. Fig. 4-3 visualizes this process of calculating G_{ij} for a set of data points that encapsulate a single partition of the state space.

4.3.1 Convergence

Consider the center of each partition, $\bar{x}_p = \int_{X_p} x dx / \Delta v_p$, and the distance between $\bar{x}_p$ and each point, $x[k_p]$:

$$\Delta x[k_p] = \bar{x}_p - x[k_p] \tag{4.19}$$

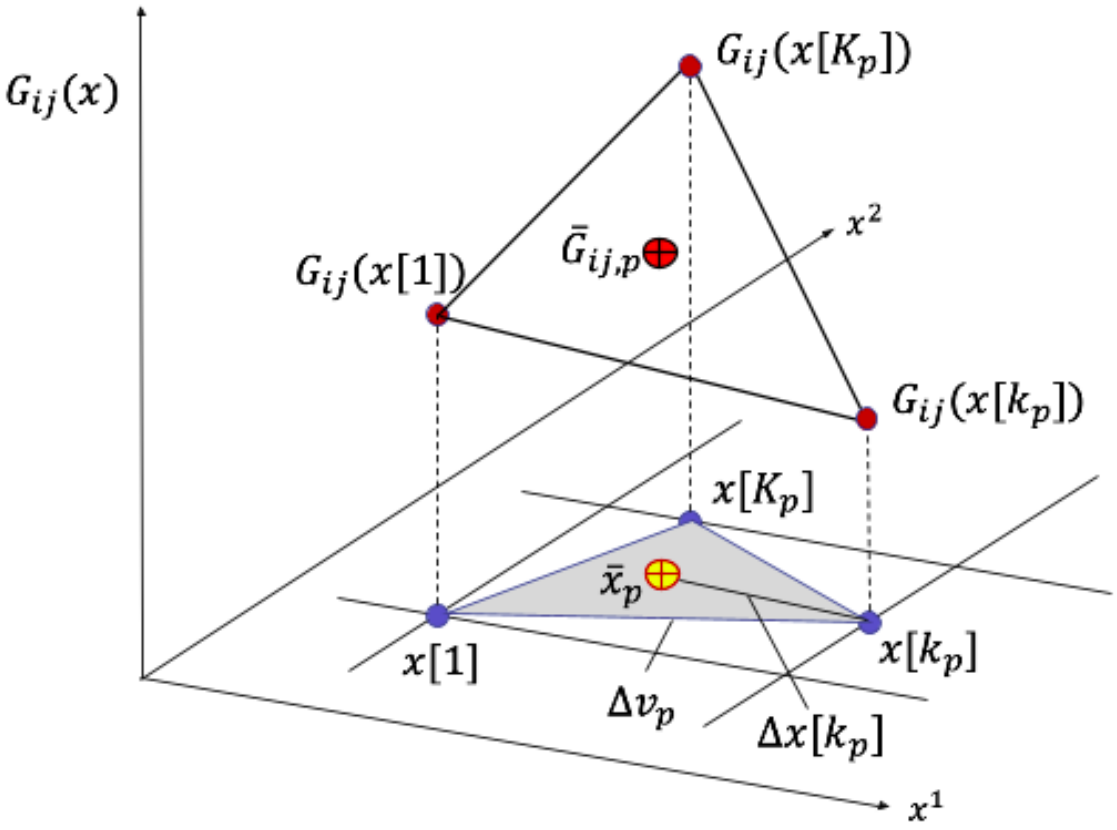

Figure 4-3: Visualization of the integrand calculation process. The volume of the partition is encapsulated by the data points is denoted as Δv_p. With this grouping, the value of G_{ij} is calculated for each point and the average among this group is computed, $\bar{G}_{ij,p}$. In turn, this value, weighted by the volume of this partition, is summed across other partitions (not shown) to approximate the value of the element R_{ij}.

See Fig.4-3. The maximum distance from the center of the partition to each point that makes up the partition is

$$|\Delta x_p| = \max\{|\Delta x[1]|, \ldots |\Delta x[k_p - 1]|, |\Delta x[k_p]|\} \tag{4.20}$$

Consider a sequence of refining the approximate inner product integral $\hat{R}_{ij}$ by increasing data points N. We can show that, as the number of partition P tends infinity and the maximum subintervals $|\Delta x_p|$ approach zero, the approximate inner product integral $\hat{R}_{ij}$ converges to its true integral.

$$R_{ij} = \lim_{\substack{P\to\infty \\ |\Delta x_p|\to 0}} \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} G_{ij}(x[k_p])\Delta v_p \tag{4.21}$$

$$Q_{ij} = \lim_{\substack{P\to\infty \\ |\Delta x_p|\to 0}} \sum_{p=1}^{P} \frac{1}{K_p} \sum_{k_p=1}^{K_p} F_{ij}(x[k_p])\Delta v_p \tag{4.22}$$

This formulation takes the form of weighted sums, specifically Riemann sums. This summation is formulated as a sequence that can be refined, where the maximum subintervals, in this case Δx_p, approach 0. Given functions that are bounded and continuous over the subdomain of interest, sequences of this form are known to have a common limit and thus converge upon refinement to the Riemann integral value over that subdomain, according to Numerical Integration theory [14, Section 1.5].

Notably, this use of the Riemann integral differs from the use of Lebesgue integrals which are used in Hilbert spaces. As the method is an application of numerical integration, the inability to generalize to all functions that belong to Hilbert spaces is a known limitation. However, one should not consider this as overly restrictive given that the observable functions chosen in state of the art methods are Riemann integrable.

4.3.2 Algorithm

In the prior section, integrals (4.15) and (4.17) are presented as summations over partitions. This computation can be streamlined by converting the summations over

partitions to the one over nodes. Consider node 3 associated to data point x_3 in Fig.4-2, for example. This node is an apex of the 5 surrounding triangles. This implies that integrand $G_{ij}(x)$ is calculated or recalled 5 times in computing (4.15) and (4.17). This repetition can be eliminated by computing volume Δv_k associated to node k, rather than partition $p : \Delta v_p$. Namely, we compute

$$\Delta v_k = \sum_{p=1}^{P} \frac{\Delta v_p}{K_p} I(k,p) \tag{4.23}$$

where $I(k,p)$ is a membership function that takes value 1 when node k is an apex of triangle p, that is, node k is involved in partition p. Using this volume as a new weight we can rewrite (4.15) and (4.17) to be

$$\hat{R}_{ij} = \sum_{k=1}^{K} G_{ij}(x[k]) \Delta v_k \tag{4.24}$$

$$\hat{Q}_{ij} = \sum_{k=1}^{K} F_{ij}(x[k]) \Delta v_k \tag{4.25}$$

Using this conversion, the computation can be streamlined and cleanly separated into three steps, as shown by pseudo-code in Algorithm 1. The steps are: (1) *Graph Creation:* data are connected to create partitions of the domain using a mesh generator: lines 3 to 8, (2) *Weighting Calculation:* calculation of the weights for each data point: lines 10 to 17, and (3) *Matrix Calculation:* the calculation of the R and Q matrices to find the matrix A, lines 19 to 21.

4.4 Experiments

In this section, the DDE algorithm is implemented for the sake of evaluating its validity and comparing its modeling accuracy to EDMD. Consider a 2nd order nonlinear system consisting of a pendulum with a nonlinear damper. See Fig. 4-4. The pendulum also bounces against walls with nonlinear compliance. The state variables for

this system are

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \tag{4.26}$$

and the equation of motion can be written as:

$$\ddot{\theta} = -sin(\theta) + F_k + F_c \tag{4.27}$$

where F_k and F_c are wall reaction moment and damping moment, respectively,

$$F_k = \begin{cases} -\text{sign}(\theta)\, k(|\theta| - \frac{\pi}{4})^2 & \text{if } |\theta| \geq \frac{\pi}{4} \\ 0 & \text{if } |\theta| < \frac{\pi}{4} \end{cases} \tag{4.28}$$

$$F_c = -\text{sign}(\dot{\theta})\, c\, \dot{\theta}^2 \tag{4.29}$$

Algorithm 1: Algorithm for DDE in pseudocode

Input:
D_N, D_N^f
Output:
P: p, K: k, R, Q A
Graph Creation:
for x_i *in* D_N *and corresponding* $f(x_i)$ *in* D_N^f **do**
    Create node, k
    Assign node attributes for current data point $k.x_t = x_t$ and $k.x_{t+1} = x_{t+1}$
    Append node to node list K.
then
Weighting Calculation:
Create list of triangles, P using Delaunay Triangulation on node list K using $k.x_t$
for p *in* P **do**
    Calculate volume, V, of p
    for k *corresponding to* K_p **do**
        Update volume of each node $k.\Delta v_k = k.\Delta v_k + \frac{V}{n+1}$
then
Matrix Calculation:
Find R and Q via (4.24) and (4.25)
Find $A = QR^{-1}$

Notation	Definition
x	state vector
x_t	an initial state drawn from the dataset D_N
x_{t+1}	x_t at next time step, drawn from D_N^f
D_N	a set of states
D_N^f	a set corresponding to D_N at the next time step
p	a partition formed from a set of $n+1$ data points
P	list of partitions
k	a node with attributes: x_t, x_{t+1}, R, Q, V
K	a list of nodes equivalent
A	the linear dynamic matrix

Table 4.1: Notation and definitions for variables indicated in different parts of this paper.

where $k = 200$ and $c = 1$. We present a two part numerical experiment for this system. The first experiment regards variations in dataset size and distribution, and the second experiment varies the usage of observable functions.

4.4.1 Dataset Variations

The datasets tested are of three types:

1. **Uniform**: These datasets are composed of a rectangular dynamic range which is sampled uniformly, like an evenly divided grid. The range varies from $\theta = [-0.8, 0.8]$ and $\dot{\theta} = [-2, 2]$, where the mass can hit the walls and the damping can vary from 0 to a significant value. See Fig. 4-4-(b), (c).

2. **Gaussian**: Data points are sampled with a finite-support Gaussian distribution. We define sampling from a finite-support Gaussian as first sampling from a Gaussian function, and then discarding all points outside of a pre-defined dynamic range. The data are distributed non-uniformly with their highest density at the peak of the Gaussian placed at diverse locations. Each dataset contains 100 data points uniformly distributed along the boarder of the dynamic range to guarantee the same dynamic range as the uniform datasets. The standard deviation is chosen to reliably sample within the truncated range; that is $\sigma_\theta = 0.08$, $\sigma_{\dot{\theta}} = 0.2$ where σ is the standard deviation.

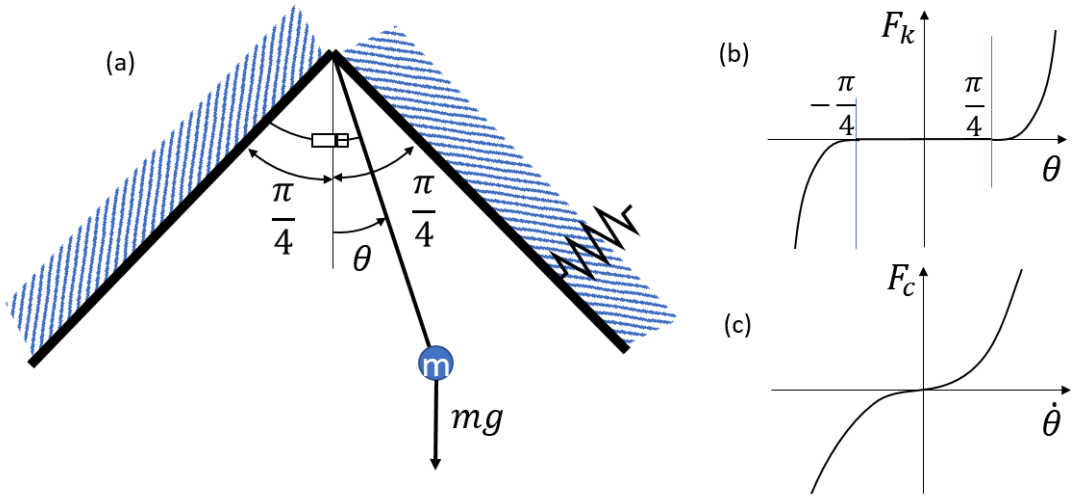

Figure 4-4: Diagram of pendulum with with walls. (a) depicts the range of the pendulum where the walls are equally angularly displaced from the vertical. (b) depicts the forces exerted on the pendulum due to contact with walls. (c) depicts the damping force exerted onto the pendulum.

3. **Trajectories**: These datasets are composed of trajectories, beginning from 100 initial conditions that are simulated forward the same number of time steps. The dynamic range of this dataset differs from the two other dataset types.

The inputs of each dataset are visualized in Figs. 4-5, 4-6, 4-7, and4-8. Corresponding graphs are presented in Fig. 4-9, 4-21, Fig. 4-19, and 4-20.

The models constructed for DDE and EDMD use the same observable functions. The observable functions chosen are two dimensional radial basis functions (RBFs), uniformly distributed between the maximum and minimum values of each state variable in their respective dataset, and the state variables. The total order of the system is 27th order with 25 RBFs and 2 state variables.

A trajectory dataset graph is generated using Delaunay Triangles in DDE, shown in Fig. 4-9.

The accuracy of the models is tested through calculating sum of squared errors (SSE) for one-step ahead predictions over the dynamic range of the datasets. These error values are calculated for a uniform grid of points, similar to that used in the uniform datasets. A visualization of the SSE is plotted in Fig. 4-10 and in Fig. 4-12.

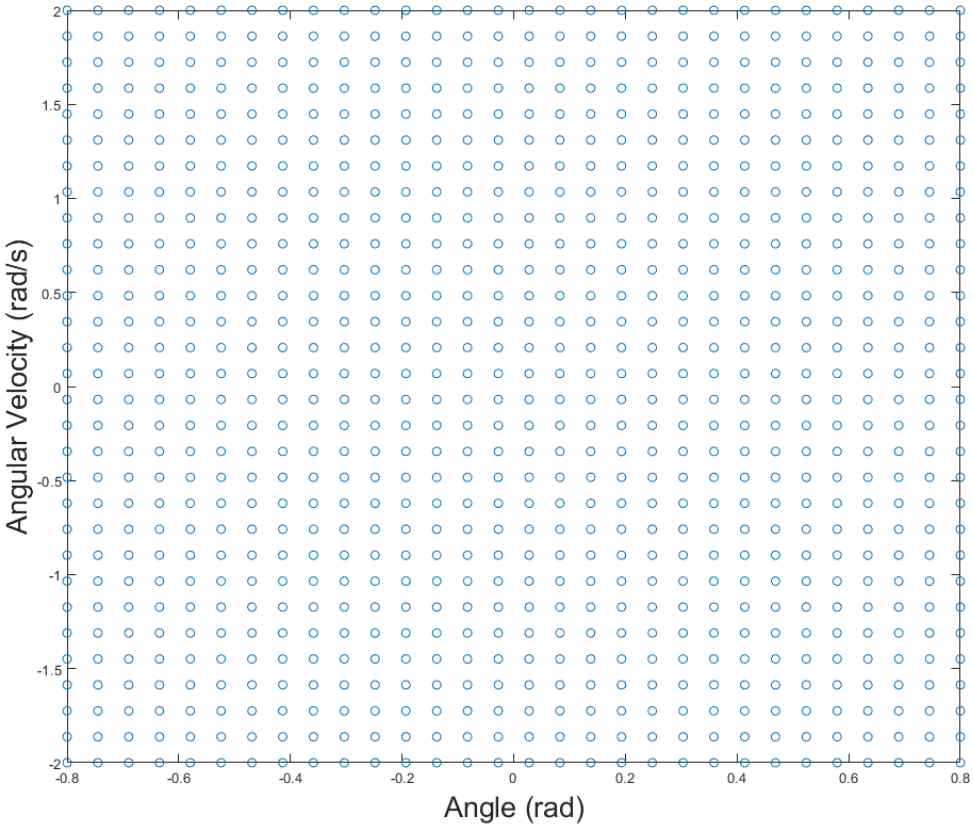

Figure 4-5: Example of data belonging to the uniform dataset with 900 data points for all snapshots belonging to x_t.

Table 4.2: Sum of Squared Errors over dynamic range with varying dataset sizes.

Dataset Size	Total SSE EDMD / DDE	SSE Variance EDMD / DDE
Uniform Datasets		
900	19.470 / **17.167**	**0.0094** / 0.0097
2500	17.995 / **16.471**	**0.0095** / 0.0103
10000	17.010 / **16.184**	**0.0099** / 0.0105
22500	16.698 / **16.133**	**0.0101** / 0.0105
Trajectory Datasets		
1000	56.532 / **33.392**	0.0349 / **0.0193**
2500	33.330 / **25.064**	0.0200 / **0.0130**
5000	31.690 / **25.099**	0.0195 / **0.0129**
10000	30.184 / **25.101**	0.0186 / **0.0129**
25000	29.380 / **25.106**	0.0181 / **0.0129**

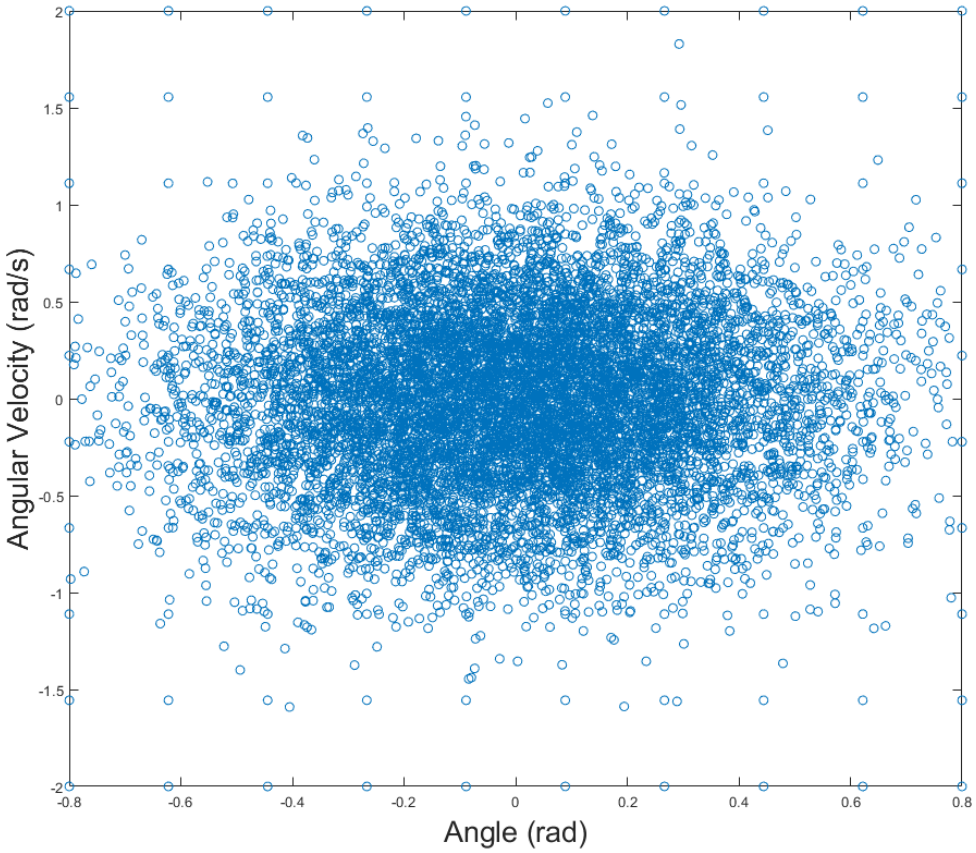

Figure 4-6: Example of data belonging to the Gaussian dataset that is centered at the origin with 10000 data points for all snapshots belonging to x_t.

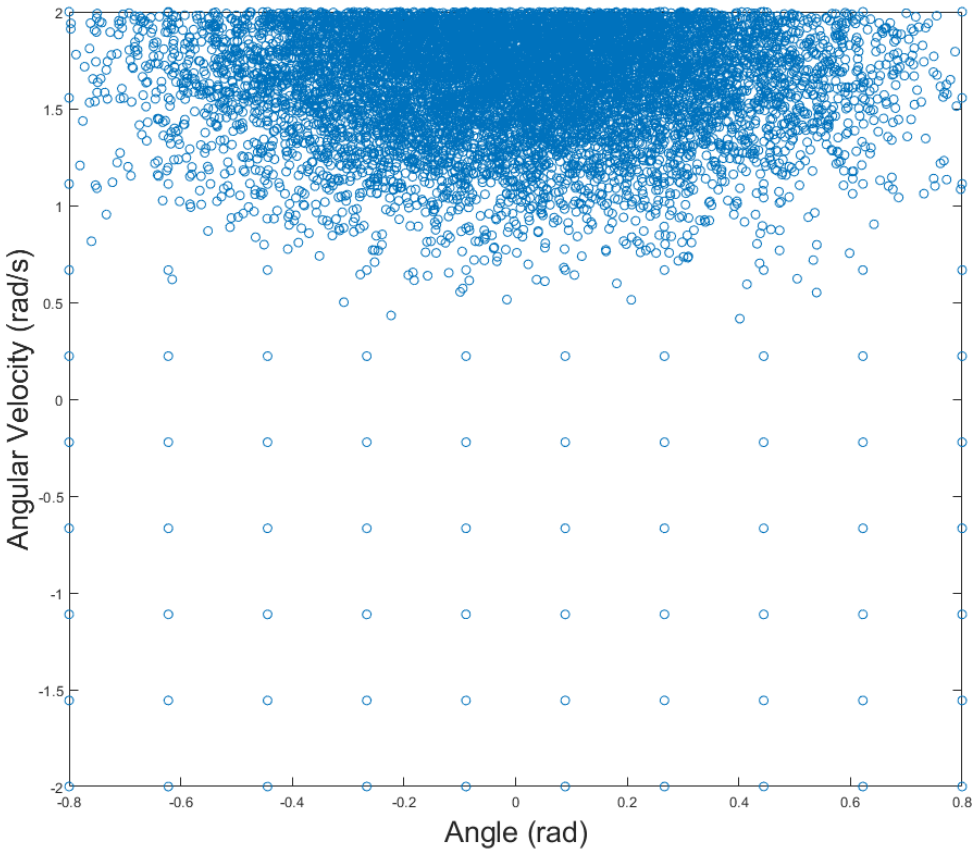

Figure 4-7: Example of data belonging to a Gaussian dataset that is centered away from the origin with 10000 data points for all snapshots belonging to x_t.

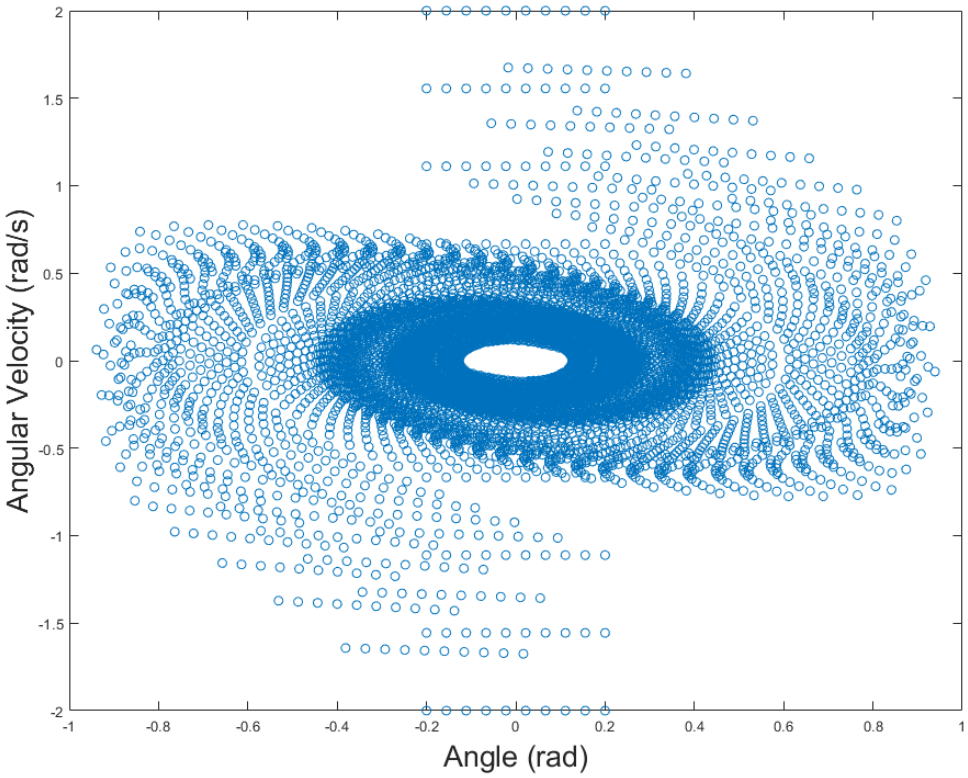

Figure 4-8: Example of data belonging to the trajectory dataset with 10000 data points for all snapshots belonging to x_t.

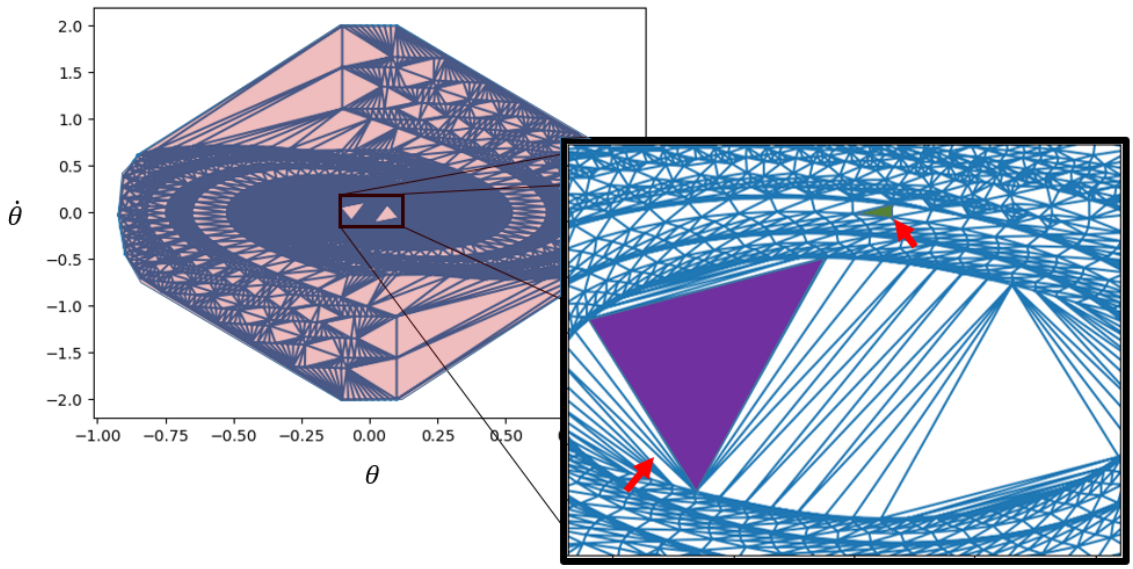

Figure 4-9: Graph of connections for a trajectory dataset composed of 10000 points formed when utilizing the data-driven direct encoding method. The lightly red shaded region denotes the dynamic range. In the zoomed-in image, the difference in volumes associated to different data points can be observed. Noting the difference in size of the purple triangle versus the green triangle.

The results of these calculations are shown in Table 4.2 and 4.3. In the computation, the dynamic range is discretized, and the SSE value of each point is summed. For the Gaussian datasets, the test is run for eight iterations of each dataset to account for randomness and the average result is noted.

To demonstrate the convergence of the parameters based on numerical integration, Figs. 4-13 - 4-18, illustrate the parameter values for various elements of the Q matrix used in DDE as a function of dataset size for each of the different dataset types.

4.4.2 Observable Function Variation

The second experiment varies the number of observable functions selected, thus increasing the order. In this experiment, the number of RBFs is varied through uniformly increasing the density of the centers of the function over the range of the dataset. The results are noted in Table 4.4. In the table, the number detailing number of observables is the number of functions including the state variables. It is shown

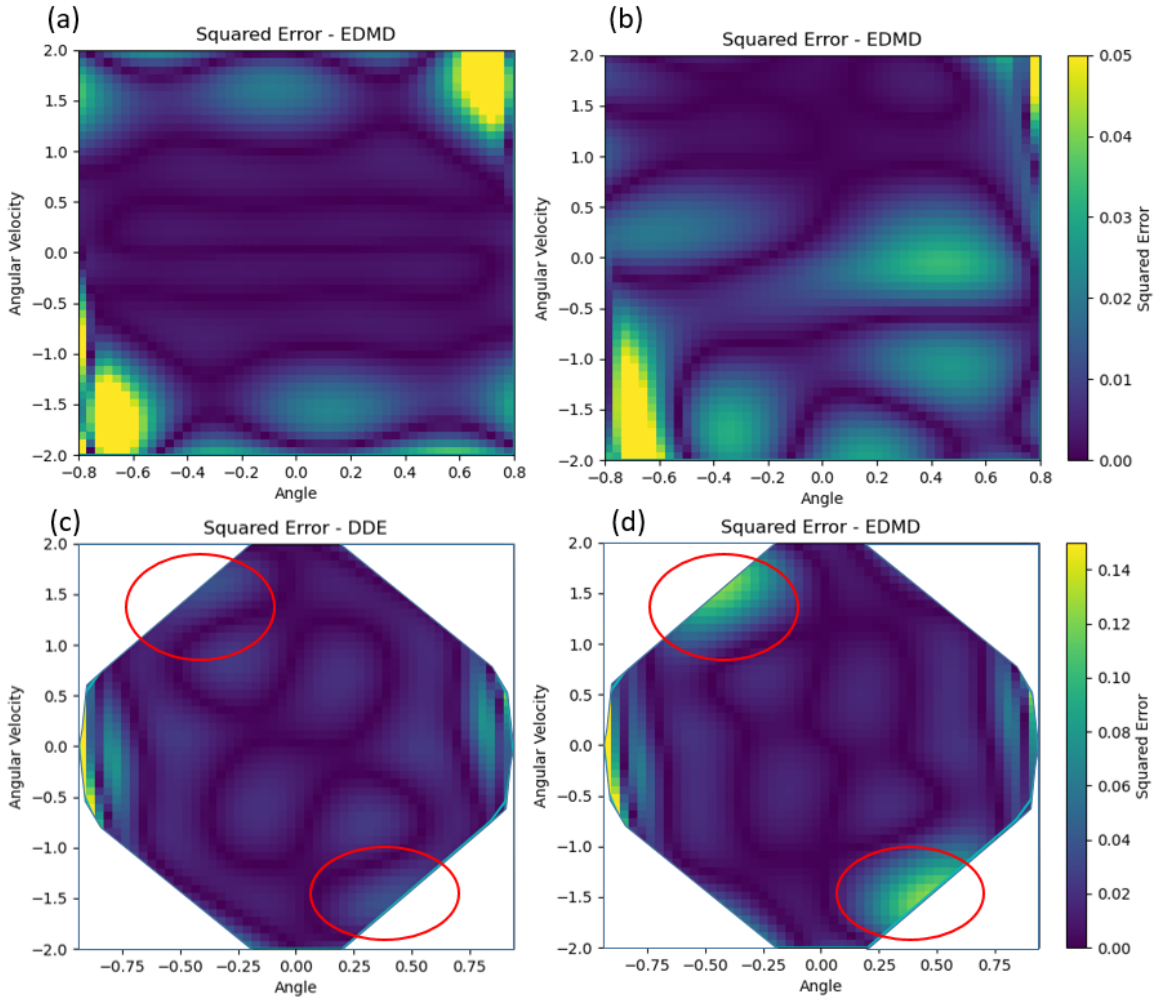

Figure 4-10: Sum of Squared Error plots for various datasets. (a) and (b) are EDMD models for Gaussian datasets; (a) uses a dataset that is centered at $[0, 0]$ and (b) uses a dataset that is centered at $[0, 2]$. (c) DDE and (d) EDMD models using the trajectory dataset composed of 2500 data points and using 25 RBFs. Circled in red are the regions with greatest variation in SSE between models. Only the dynamic range is shown in all plots.

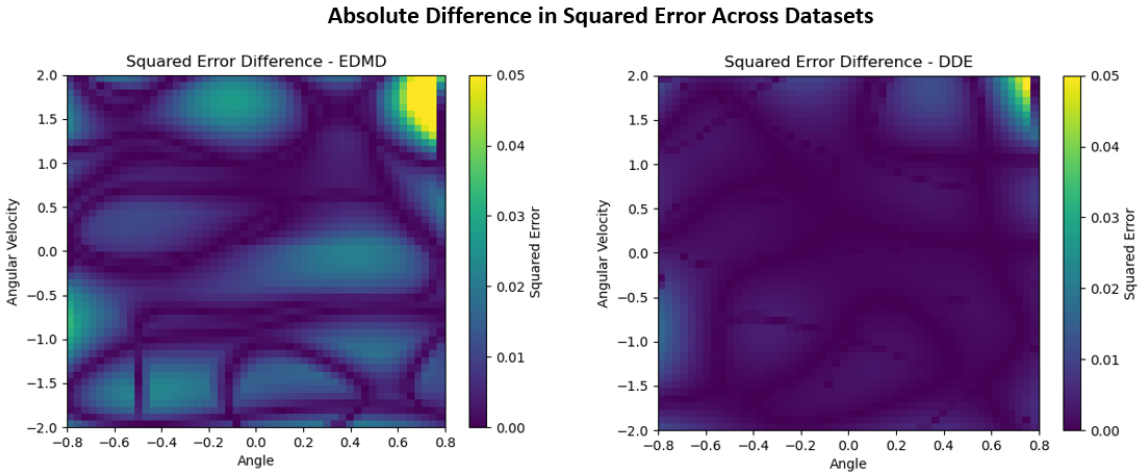

Figure 4-11: Absolute difference in Sum of Squared Error across two different models. These heatmaps are formed by generating models for two different Gaussian datasets, one with mean centered at the origin and the other centered at $[0, 2]$, and calculating the absolute difference in SSE. On the left is the difference for an EDMD model, which is notably highlighted throughout the dynamic range. On the right is the difference in SSE for the DDE model, which is in general nearly zero, indicating a consistent model despite large differences in data.

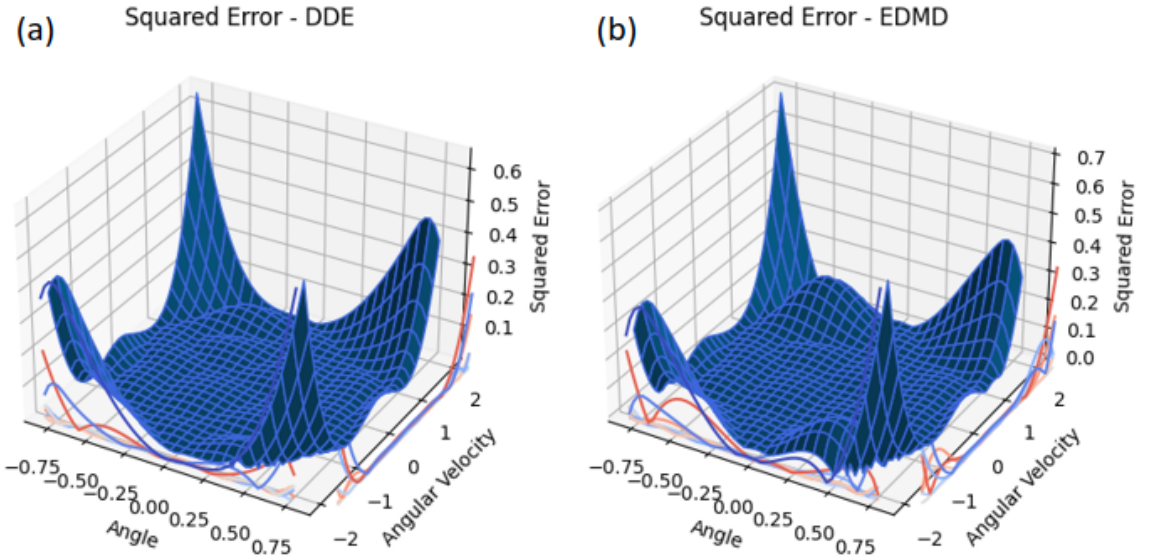

Figure 4-12: Sum of Squared Error plots for a trajectory dataset consisting of 2500 data points. (a) corresponds the DDE model, and (b) corresponds to the EDMD model. Provides an alternative view of Fig. 4-10(c) and (d)

Table 4.3: Sum of Squared Errors over dynamic range for Gaussian distributions with different centers. Centers of the distribution are noted above each column.

Dataset Size	Total SSE Center: $[0, 0]$ EDMD / DDE	 Center: $[0.8, 0]$ EDMD / DDE	 Center: $[0, 2]$ EDMD / DDE
	Gaussian Datasets		
1000	25.565 / **24.377**	25.161 / **22.98**	24.338 / **22.445**
2500	24.991 / **23.739**	25.421 / **22.747**	24.221 / **21.590**
5000	24.662 / **23.518**	26.805 / **22.415**	23.914 / **21.195**
10000	24.395 / **23.085**	28.424 / **21.825**	25.770 / **21.052**
25000	25.219 / **22.167**	30.788 / **21.687**	26.941 / **20.704**

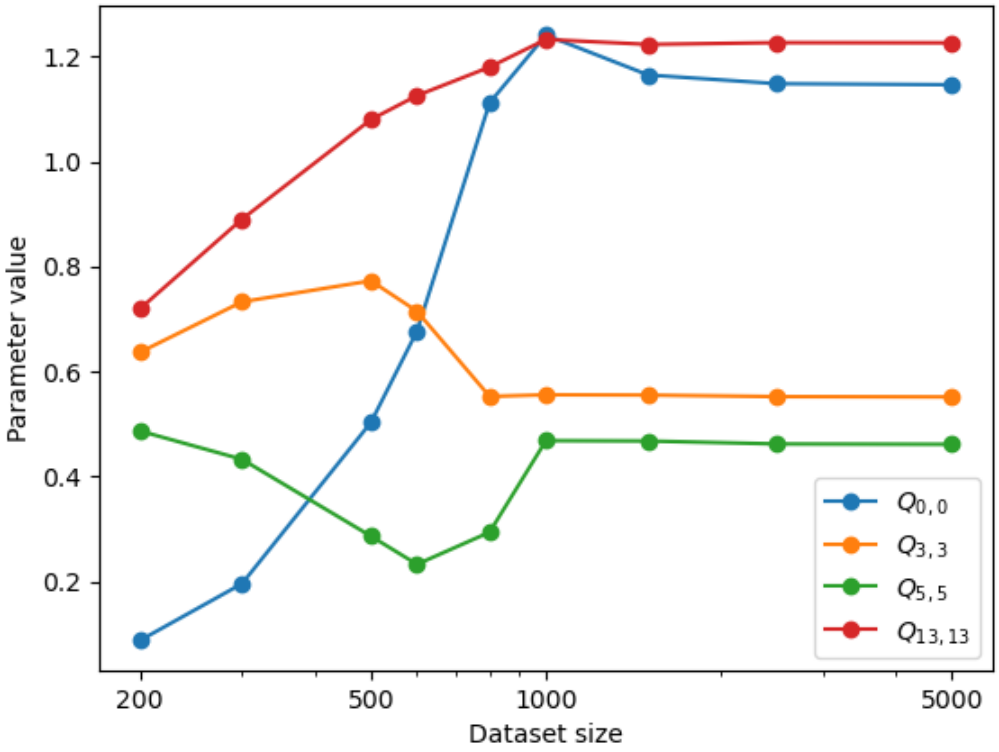

Figure 4-13: Change in values of diagonal elements in the Q matrix of DDE with respect to trajectory dataset size. The specific elements shown correspond to a state variable, $Q_{0,0}$, and three RBF functions of varying distances away from the equilibrium.

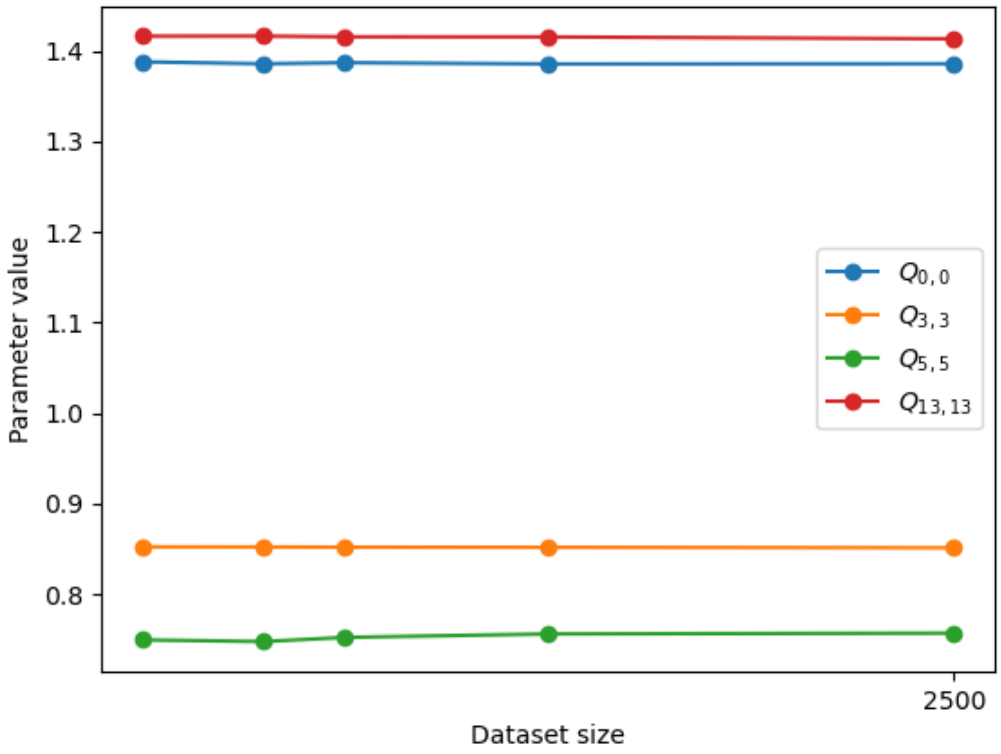

Figure 4-14: Change in values of diagonal elements in the Q matrix of DDE with respect to gaussian dataset size. The specific elements shown correspond to a state variable, $Q_{0,0}$, and three RBF functions of varying distances away from the equilibrium.

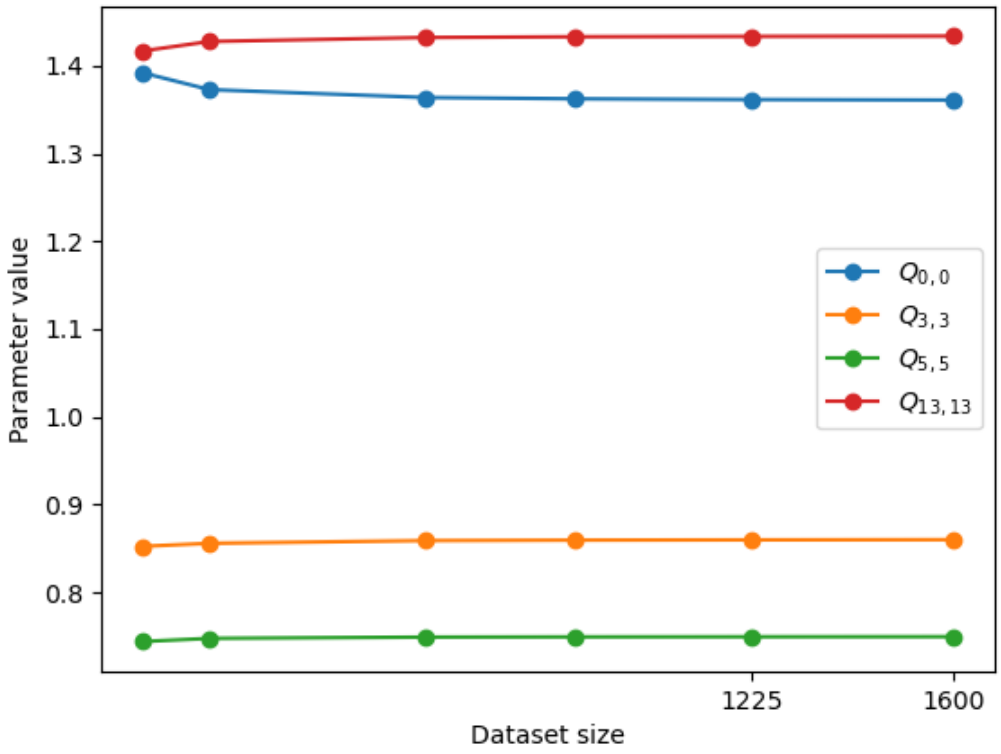

Figure 4-15: Change in values of diagonal elements in the Q matrix of DDE with respect to uniform dataset size. The specific elements shown correspond to a state variable, $Q_{0,0}$, and three RBF functions of varying distances away from the equilibrium.

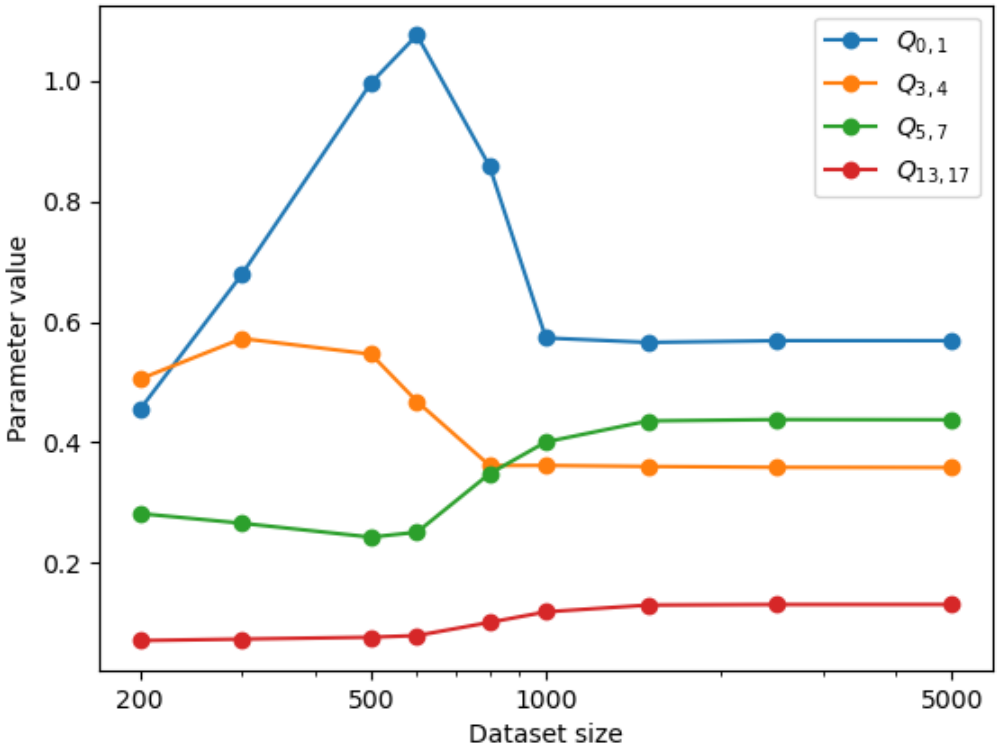

Figure 4-16: Change in values of off diagonal elements in the Q matrix of DDE with respect to trajectory dataset size.

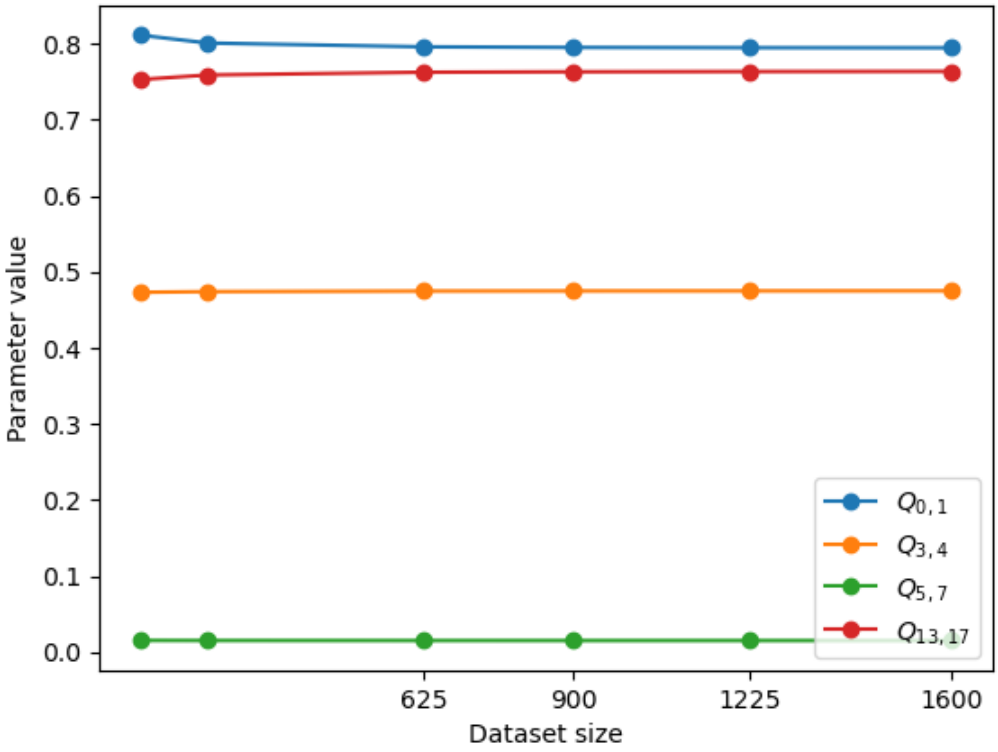

Figure 4-17: Change in values of off diagonal elements in the Q matrix of DDE with respect to uniform dataset size.

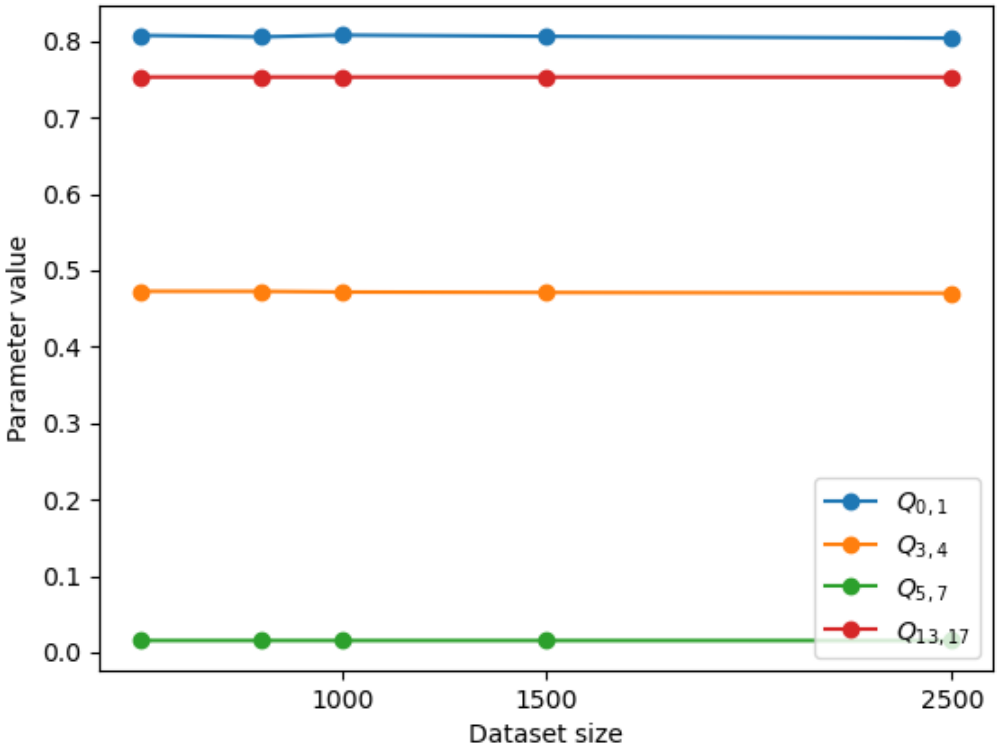

Figure 4-18: Change in values of off diagonal elements in the Q matrix of DDE with respect to Gaussian dataset size.

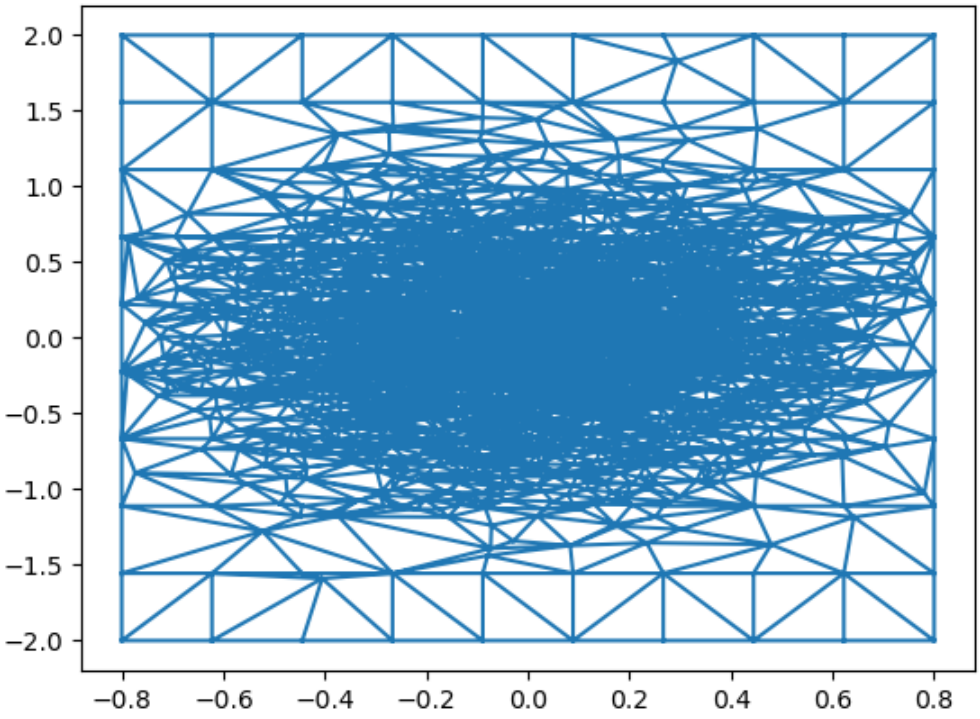

Figure 4-19: Graph of connections for the Gaussian dataset with a center at the origin formed when utilizing the data-driven direct encoding method.

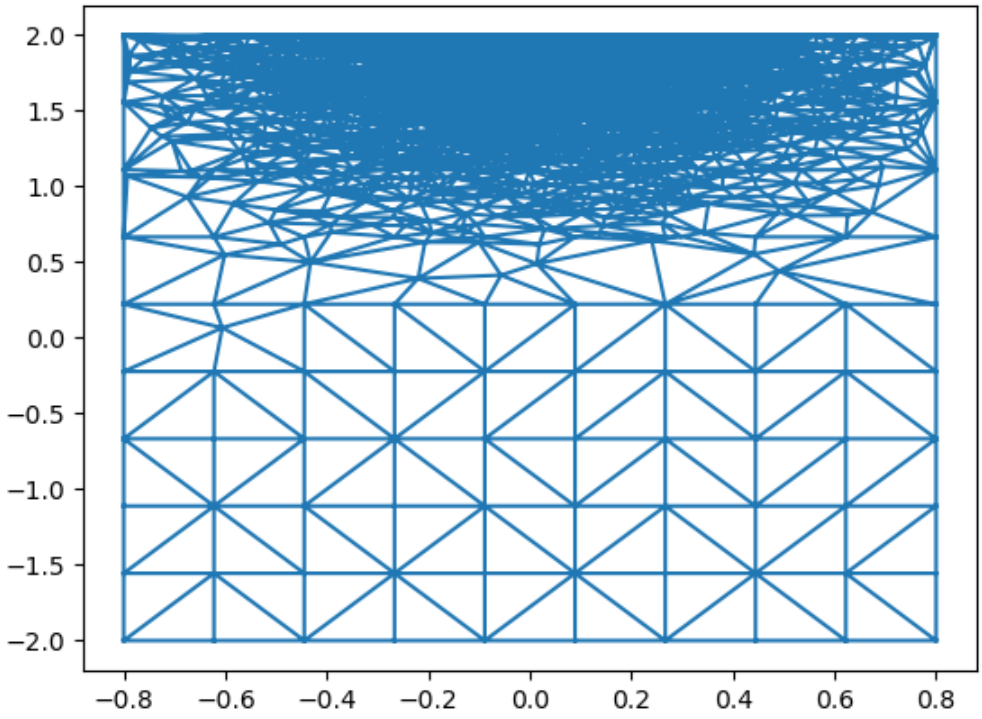

Figure 4-20: Graph of connections for the Gaussian dataset with a center away from the origin formed when utilizing the data-driven direct encoding method.

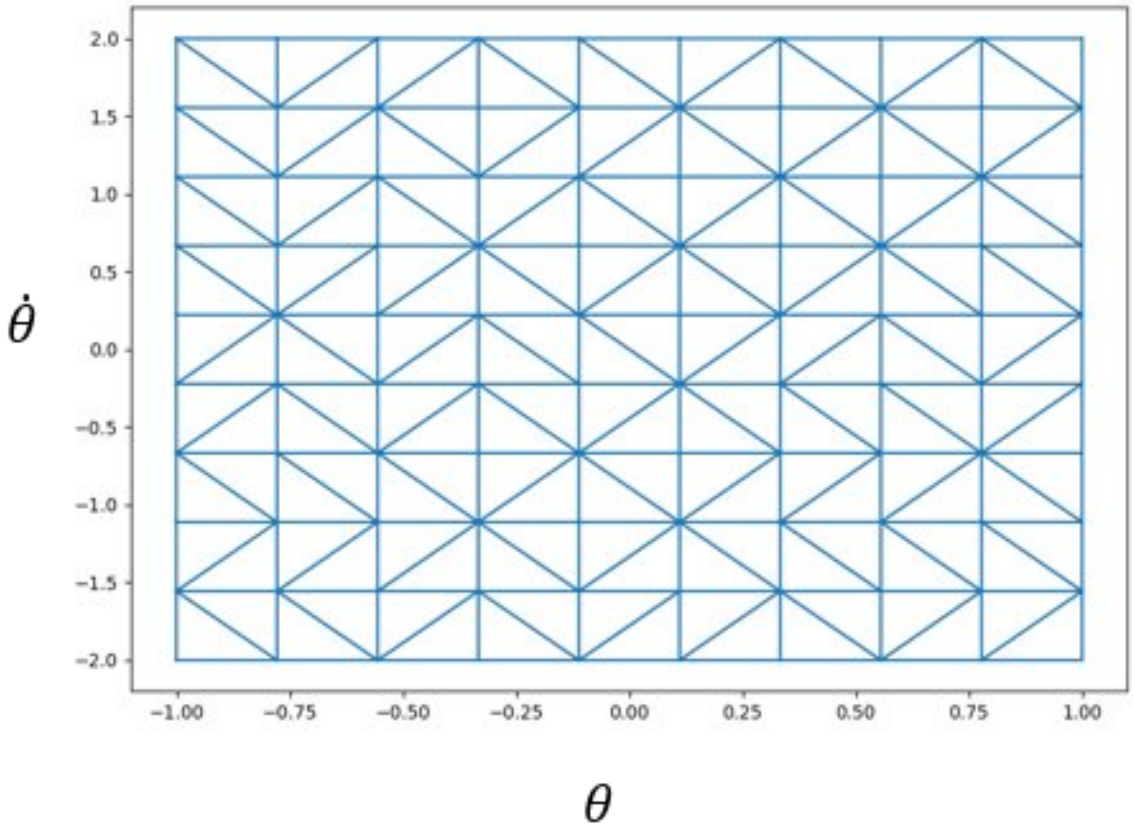

Figure 4-21: Graph of connections for the uniform dataset formed when utilizing the data-driven direct encoding method.

Table 4.4: Sum of Squared Errors over dynamic range with varying order of lifted linear models.

# Observables	Total SSE	
	EDMD	DDE
Trajectory Dataset, 5000 points		
27	31.690	**25.099**
51	36.657	**21.637**
83	28.437	**13.613**

as a bar chart in Fig. 4-22.

4.4.3 Discussion

From these results we can make the following observations.

All the numerical experiments show that DDE outperforms EDMD in total SSE. For uniform datasets, both models are nearly equivalent, though DDE has slightly lower SSE in all cases, shown in Table 4.2. This result is expected as all data points are weighted equally in a uniform distribution.

EDMD models exhibit significantly different distributions of prediction error, depending on dataset distribution. In Fig. 4-10-(a), the EDMD model learned from a dataset with high density near the origin produces a prediction error distribution that is low in accuracy in the top-right and the bottom-left corners of the dynamic range. In contrast, Fig.4-10-(b) illustrates that when using EDMD to learn from data with high density at $\theta = 0, \dot{\theta} = 2$, the model results in low accuracy in the lower half of the dynamic range. DDE does not exhibit this high distribution dependency and achieves lower total SSE, as shown in Table 4.3.

In the trajectory datasets in Table 4.2, DDE is not only lower in total SSE than EDMD but is also smaller in variance. Fig. 4-10-(c) shows that DDE has a uniformly low error distribution across the dynamic range, while EDMD in Fig.4-10-(d) has two regions, as circled in the figure, with significantly larger error. In the figure, the error plots over the dynamic range of the data are similar except for, as circled in

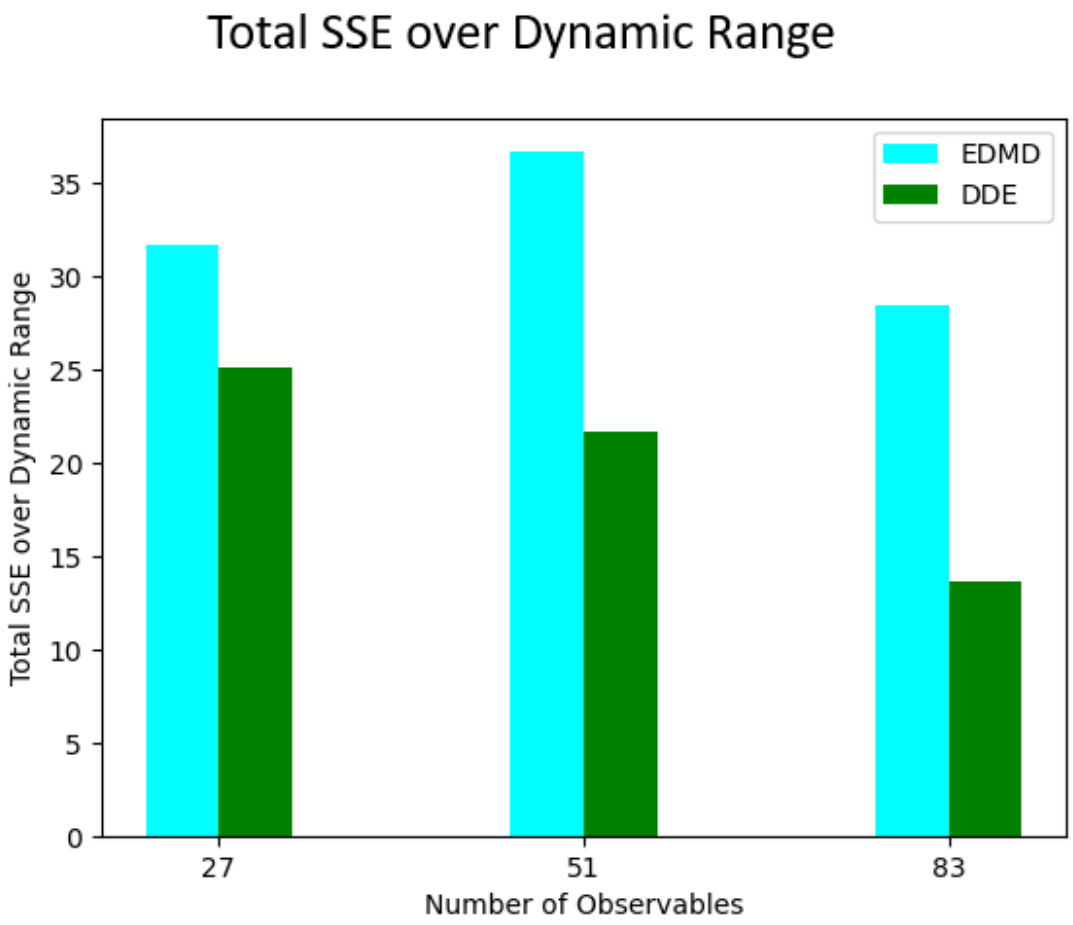

Figure 4-22: Differences in Total SSE over the dynamic range for a Trajectory dataset of 5000 data points, with varying numbers of Radial Basis Functions.

the figure, two regions on which have a significant difference between models. These regions coincide with sparsity in the dataset, providing evidence of EDMD's bias towards regions of high data density and explaining the difference in performance between models for the non-uniform datasets.

In the trajectory datasets, the total SSE converges for DDE beginning with small dataset sizes. This result implies that the elements in the R and Q matrices of DDE, that is, the inner product integral computations, converge as the data size and the data density increase. This convergence is confirmed in Figs. 4-13-4-18, where several elements of the Q matrix are plotted against the data size. These elements are representative of the elements not pictured in the R and Q matrices.

The second experiment, regarding variations in observable function numbers demonstrates the effect of DDE remains even for significant increases in observable functions, referring to Table 4.4. In all cases tested, DDE significantly outperforms EDMD over the dynamic range, as expected for a non-uniform dataset.

4.5 Conclusion

In this chapter, a new data-driven approach to generating a Koopman linear model based on the direct encoding of Koopman Operator (DDE) is presented as an alternative to dynamic mode decomposition (DMD) and other related methods using least squares estimate (LSE). The major contributions include: 1) The analytical formula of Direct Encoding is converted to a numerical formula for computing the inner product integrals from given data; 2) An efficient algorithm is developed and its convergence conditions to the true results are analyzed; 3) Numerical experiments demonstrate a) greater accuracy compared to EDMD, b) lower sensitivity to data distribution, and c) rapid convergence of inner product computation. However, the current method does not rigorously allow for control, and as such is a future direction for exploration.

Chapter 5

Deep Learning Approaches

In this chapter, we discuss the use of modern Deep Learning methods used with the Koopman Operator framework. We begin by discussing the recent work that involves the use of Deep Learning to formulate a lifted linear system, specifically to learn observable functions. Afterwards, the use of DDE is demonstrated on a 6th order nonlinear system, demonstrating the ability to accurately model the system with a low number of observables. Then, the final contribution of this thesis, a novel training method for producing observable functions, is presented. This training method is utilized in tandem with DE, demonstrating high accuracy for a system with both stable and unstable subspaces, and the capacity to illustrate the boundary between subspaces.

5.1 Introduction

Linearization methods, such as those based on the Koopman Operator, have been used to transform nonlinear systems to linear models. The theory states that the linear model becomes exact in modeling a nonlinear system as the order of the linear model approaches infinity, though some nonlinear systems have finite order representations in lifted space [8, 40]. The Koopman Operator models are generally constructed through data-driven methods such as Extended Dynamic Mode Decomposition (EDMD) [50]. These Koopman-DMD methods have been applied to non-

autonomous systems to construct linear dynamic models that allow us to apply various linear control methods, such as linear model predictive control (MPC), to nonlinear control systems [25].

A key component necessary to constructing a Koopman Operator-based linear model is selection of the observable functions that lift the state space. Prior work has studied the use of various function families as observables, such as polynomial basis functions, radial basis functions and time delays [22, 38, 47] . There have also been formulas created for algorithmically determining useful observables based on the dataset [13, 36]. Modern machine learning techniques have been applied to learn observable functions with significant success [19, 34, 51].

However, it can be difficult to formulate accurate approximations of the Koopman Operator for nonlinear systems that produce both stable and unstable trajectories. A passive dynamic walker, for example, is essentially an unstable system, but it can walk stably if it starts within a stable region [39]. When concerned with only the stable regions of a nonlinear system, methods have been developed to construct stable Koopman models from unstable data-driven models for systems that are known to be stable [6, 15, 18, 35]. These works provide methods to construct stable Koopman models from unstable Koopman models. However, these methods are not applicable when needing to predict stable trajectories for a system with unstable regions. A key difficulty is to capture a proper dataset that represents diverse behaviors involved in stable and unstable trajectories. Because of the nature of unstable trajectories, a bias towards the unstable modes can often occur when creating the Koopman model.

Prior work discusses the potential for Koopman Operator models to describe unstable subspaces [40]. This paper presents a methodology for constructing an accurate Koopman model for nonlinear systems with both stable and unstable regions through separation of a lifted space into stable and unstable subspaces. Two sets of effective observables are learned separately and superposed to construct a complete model.

In addition, a method will be developed for determining a boundary between stable and unstable regions in the original state space by analyzing the Koopman Operator Model using modal decomposition, which is made possible with the effective

construction of observable functions. In the field of Deep Learning, attempts at modeling unstable systems have relied on novel loss functions, such as a time-weighted loss [41], but, in general, such methods have demonstrated little success at separating the dynamics of such systems.

The current chapter presents two significant points. The first is a demonstration of the use of DDE in coordination with Deep Koopman methods. The next is a novel training method of subspace specific observable generation (SSOG) via a neural network.

5.2 Deep Koopman with Data-Driven Encoding

The use of deep neural networks for finding effective observable functions and constructing a Koopman linear model has been reported by several groups [33, 46, 51]. This method, sometimes referred to as Deep Koopman, is effective for approximating the Koopman operator to a low-order model, compared to the use of locally activated functions, such as RBFs, which scale poorly for high-order nonlinear systems. The proposed DDE method can be incorporated into Deep Koopman, further improving approximation accuracy.

Fig. 5-1 shows the architecture of the neural network similar to the prior works [33, 46, 51]. The input layer receives training data of independent state variables. The successive hidden layers produce observable functions; these functions feed into the output layer consisting of linear activation functions. This linear output layer corresponds to the A matrix that maps the observables of the current time to those of the next time step, i.e. the state transition in the lifted space. In the Deep Koopman approach, the output layer, that is, the A matrix, is trained together with observable functions in the hidden layers. Using the observable functions learned from deep learning, the output layer weights are replaced with the A matrix obtained from DDE, captioned in Fig. 5-1.

The number of the units in the final hidden layer connected to the linear output layer is the number of observables. For this set of observable functions and data

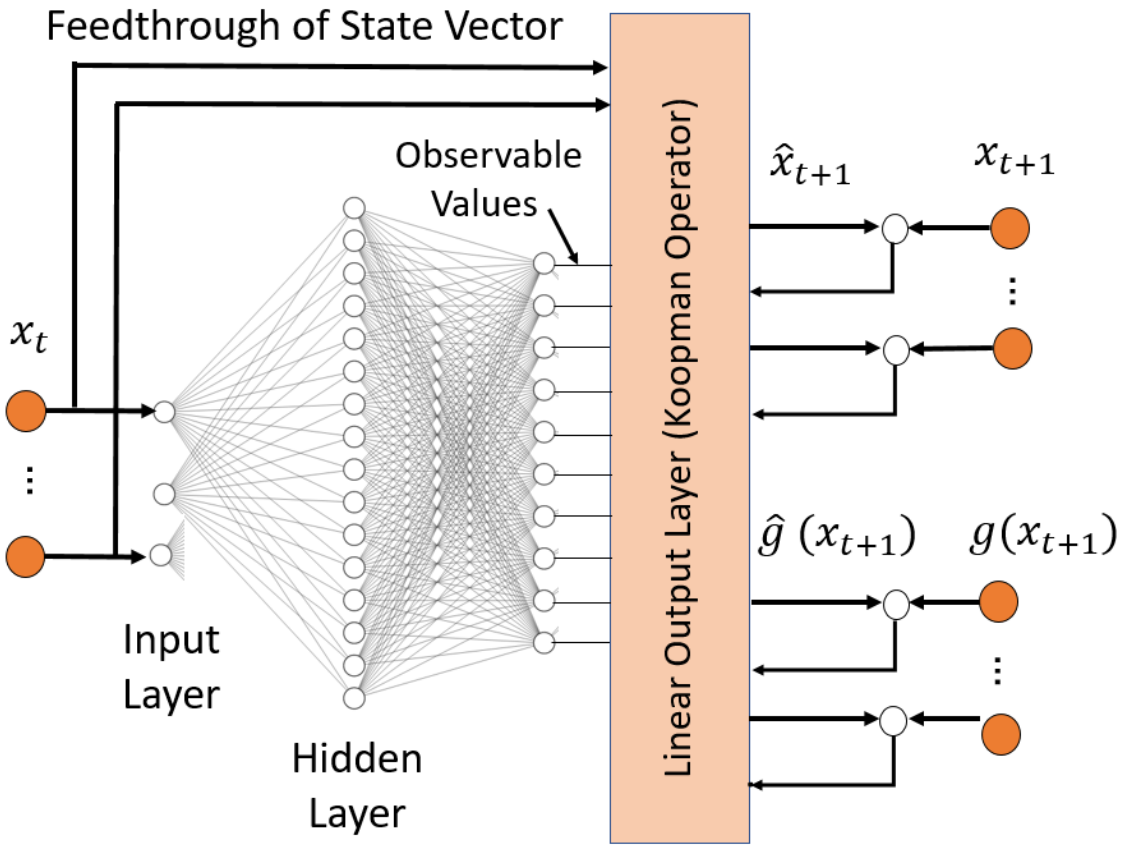

Figure 5-1: Feedforward neural network model used to generate observable functions. The final layer of the neural network is a linear layer. In the standard Deep Koopman model this remains the same after the model is fully trained. However, with the DDE model, this final layer is recalculated using DDE by taking in the dataset used to train the model.

matrices D_N and D_N^f, we can apply the DDE algorithm for finding the A matrix. The neural net training can be repeated for refining the observables after the A matrix is determined from the DDE. Furthermore, the DDE and the neural net training can be repeated multiple times in a bootstrap manner.

The trained observable functions are used for constructing a high-dimensional state vector z.

$$z = [g_1(x; w_1), \cdots, g_m(x; w_m)]^T \tag{5.1}$$

where w_k is the weights of all the hidden layer units associated to the k-th state variable in the lifted space. The effectiveness of this Deep Koopman-DDE method is applied to a multi-cable manipulation system [42]. A simplified single cable version is utilized consisting of six independent state variables. The network is constructed using PyTorch, trained with an Adam optimizer, to generate 40 observable functions. The loss function of the neural network is the mean squared error loss function

$$L_{MSE} = \frac{1}{N} \sum_{n=1}^{N} ||z_{t+1} - Az_t||^2 \tag{5.2}$$

These model concatenates the state variables of the nonlinear system, resulting in a 46th order model. Table 5.1 shows parameters used for training the Deep Koopman model. The training dataset is composed of 3000 data points drawn from trajectories. Table 5.2 compares the Deep Koopman model to the proposed model that uses DDE. Results are in terms of sum of squared error over a set of test trajectories. A significant improvement is achieved by incorporating DDE into the deep learning method.

Table 5.1: Neural network parameters and characteristics.

Parameters and Characteristics	Value
Number of Hidden layers	3
Activation Functions, Both Hidden Layers	ReLU
Width of 1st Hidden Layer	16
Width of 2nd Hidden Layer	16
Width of 3rd Hidden Layer	40
Learning Rate, α	0.01

We utilize a simplified, one winch model in this section to demonstrate the effect

Table 5.2: Average SSE prediction error for trajectories in set of test data for single winch system.

Modeling Method	1 Time Step	20 Time Steps
Deep Koopman only	0.2471	9.8610
Deep Koopman + DDE	**0.2350**	**4.1131**

of DDE with Deep Koopman. A diagram for this system is shown in Fig. 5-2. The dynamics for this system are similar, though not identical to the system introduced in Chapter 3.1.3.

The corresponding state vector for this sixth order system is

$$\mathbf{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & L_A & \dot{L}_A \end{bmatrix}^T \tag{5.3}$$

where L_A is the *unstretched* cable lengths. This unstretched cable length is defined as function

$$L_A = L_0 + r_{w_A}\theta_i$$

and the unstretched cable length velocity is the time derivative of the above function. The winches are allowed to rotate, but are fixed in position. Their rotational dynamics are defined as

$$I_A\ddot{\theta}_A = u_A - \tau_{w_A}$$

where τ_{w_A} is the torque due to the cable when in tension, and u_A is the input torque from the motor attached to the winch. This torque is defined as

$$\tau_{w_A} = r_{w_A}\mathbf{n}_{w_A} \times T_A\mathbf{n}_A$$

where r_{w_A} is the radius of the winch, and $\mathbf{n}_A$ corresponds to the unit vector in the direction of the departure point of the cable from the center of the winch. The mass

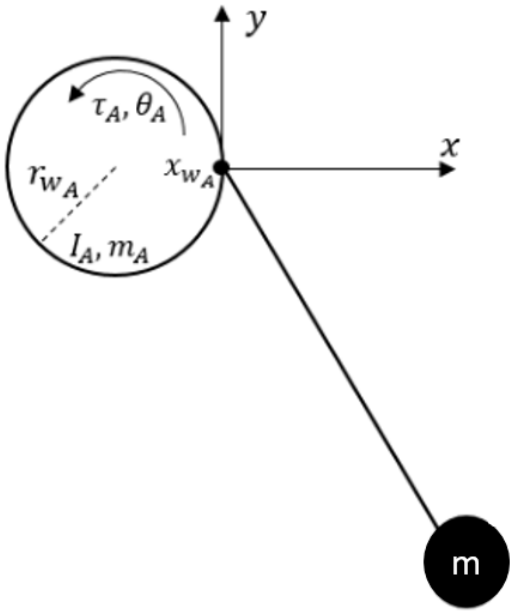

Figure 5-2: Diagram of one winch system consisting of one winch with rotational inertia, two cables, and a suspended mass. The motor that drive the winches are controlled with PD controllers that attempt to guide the unstretched cable length to specific values.

switches dynamic equations based on the conditions

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{cases} \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{if } d_A \leq 0 \\ T_A \mathbf{n}_A + \begin{bmatrix} 0 \\ mg \end{bmatrix} & \text{if } d_A > 0 \end{cases} \tag{5.4}$$

where d_A is the elongation length of cable A subtracted by its unstretched length based on the positioning of the mass relative to the respective winch.

The equations that govern the rope's extension with respect to force are nonlinear, and derived experimentally as in the prior work. The equation for the tension of the rope is

$$T_A = 1699 d_A \tag{5.5}$$

In Fig. 5-4, an example trajectory for the mass position in the horizontal direction

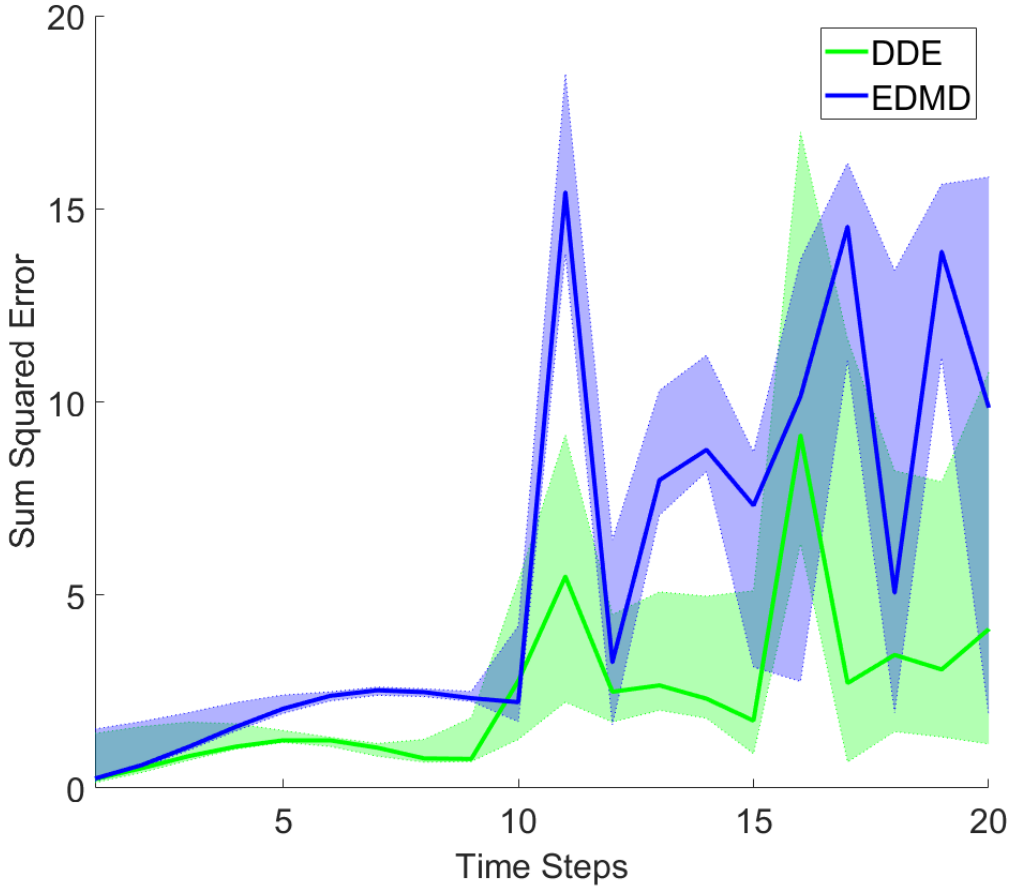

Figure 5-3: Sum of Squared Error (SSE) plot for one hundred trajectories for both EDMD and DDE models using the grid point dataset. The shaded regions represent the minimum and maximum errors and the corresponding line is the average SSE.

is shown, and in Fig. 5-3, a SSE error plot is shown presenting the SSE of one hundred trajectories represented by the shaded regions.

For the realistic experiment of applying this to the sixth order winch system, it is clear that for higher order systems that the effect of calculating the A matrix using DDE leads to improved results when compared to the EDMD approach as shown in Fig. 5-3 and 5-4. With this increase in prediction accuracy, the method presented can be deemed as effective even when applying neural networks for generation of observable functions.

Practical concerns arose from the use of Delaunay Triangulation in this experiment. Specifically, the method was found to be limiting and inconsistent for systems

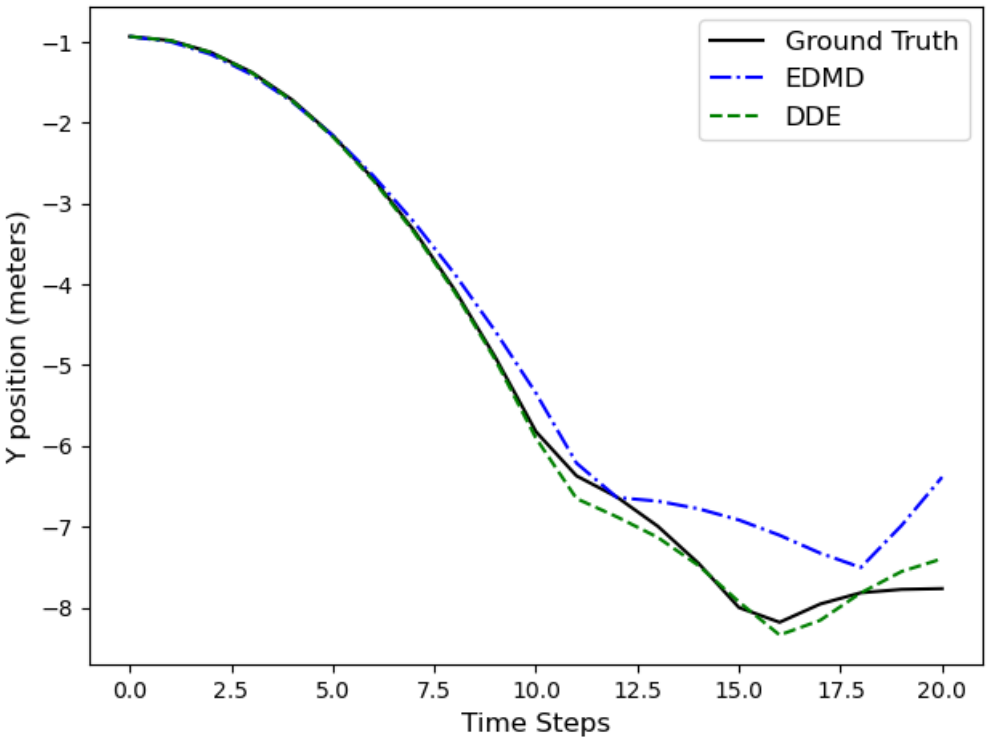

Figure 5-4: Trajectory example comparing the prediction of the EDMD and DDE models for the y position of the point mass.

that are eighth order or higher, due to computational cost and numerical instability. Alternative numerical integration approaches or alternative partitioning methods may be necessary when applying the method to higher order nonlinear systems.

5.3 Subspace Specific Observable Generation

This section presents a novel algorithm for obtaining an accurate Koopman operator model for nonlinear systems having both stable and unstable regions. The algorithm is built upon three theoretical and technical foundations.

First, it employs the Direct Encoding formula. In DMD, including its variants such as EDMD, the linear state transition matrix A is assumed to exist and is determined from data based on a Least Squares Estimation. This may cause a biased estimate, as addressed previously. In the Direct Encoding formula, however, the A matrix is determined from the inner products of observable functions and their composition with the nonlinear state function f involved in the two matrices R and Q. Because both g_i and $g_i \circ f$ are in a Hilbert space, all the inner products are guaranteed to exist and the resultant A matrix provides an exact linearization that does not depend on data. The linear model is globally valid. This Direct Encoding formula is used as a foundational framework in the new algorithm.

Second, the algorithm exploits a basic property of a linear dynamical system. If a Koopman Operator model can be constructed for a nonlinear system, then the system can be represented by

$$z_{k+1} = Az_k \tag{5.6}$$

from eq. (2.1). This linear system can be separated into its individual modes using eigendecomposition.

$$z_{k+1} = V_u D_u^t W_u^T z_k + V_m D_m^t W_m^T z_k + V_s D_s^t W_s^T z_k \tag{5.7}$$

where V, D, and W are the eigendecomposition of A, and the subscripts u, m, and

s represent unstable, marginally stable, and stable subspaces. This decomposition in the lifted space motivates us to construct observable functions that represent the individual subspaces.

However, when represented in the state space of the original nonlinear system, these subspaces may be not as simple to find the boundaries in between; an artistic visualization of such boundaries is shown in Fig. 5-6.

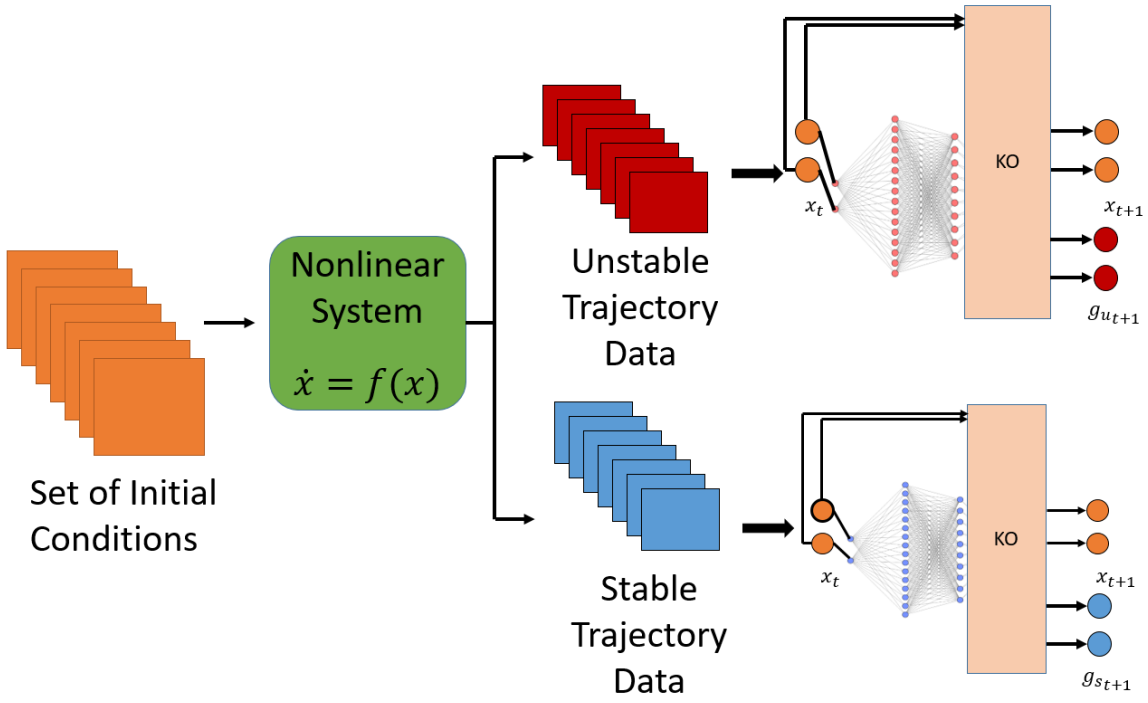

Figure 5-5: The training scheme utilized to generate the architecture in Fig. 5-7.

Third, the algorithm uses neural networks to find an effective set of observables for each of the stable and unstable subspaces. The training method, summarized in Fig 5-5, assumes that while the underlying system has unknown characteristics, the trajectory data obtained from the system can be separated into two categories: stable/marginally stable, and unstable. Specifically, an aggregate dataset is comprised of initial states X_k and the states at one time step ahead X_{k+1}. This can then be organized into

$$X = \begin{bmatrix} X_u & X_s \end{bmatrix} \tag{5.8}$$

for both X_k and X_{k+1}. These subsets of data are used to train corresponding neural

networks $G_u(x_i;\theta_i)$ and $G_s(x_i;\theta_i)$ and the weights of linear output layers A_s and A_u. The function of the neural networks is to produce observables of a given state, that is

$$G_*(x) = g_*(x) \tag{5.9}$$

and

$$z_{*_{k+1}} = A_* z_{*_k} \tag{5.10}$$

where $*$ can be replaced with s or u, representing the specific subset of data, and z is the lifted state represented by

$$z_* = \begin{bmatrix} x_* & g_*(x_*) \end{bmatrix}^T \tag{5.11}$$

The loss function utilized in training all models is

$$L(z,\hat{z}) = \frac{1}{N}\sum_{i=0}^{N}(z-\hat{z})^2 \tag{5.12}$$

After the training of these networks is completed with the trajectory dataset, the weights for the neural networks G_u and G_s no longer are updated. We then utilize the Direct Encoding method to produce a new estimation for the output layer. This new regression is computed by creating a new lifted state through concatenating the observable functions of each network with the state vector

$$z = \begin{bmatrix} x & g_u & g_s \end{bmatrix}^T \tag{5.13}$$

The observable functions produced by these networks are then concatenated as shown in Fig. 5-7. This new linear transition matrix can be recomputed utilizing the Direct Encoding method. Let $g_i(x;\theta_i)$ be the i-th observable in the form of a neural network with parameters θ_i. The inner product comprising the matrix R is give by

$$R = \{R_{ij}\} = \{\langle g_i(x;\theta_i), g_j(x;\theta_j)\rangle\} \tag{5.14}$$

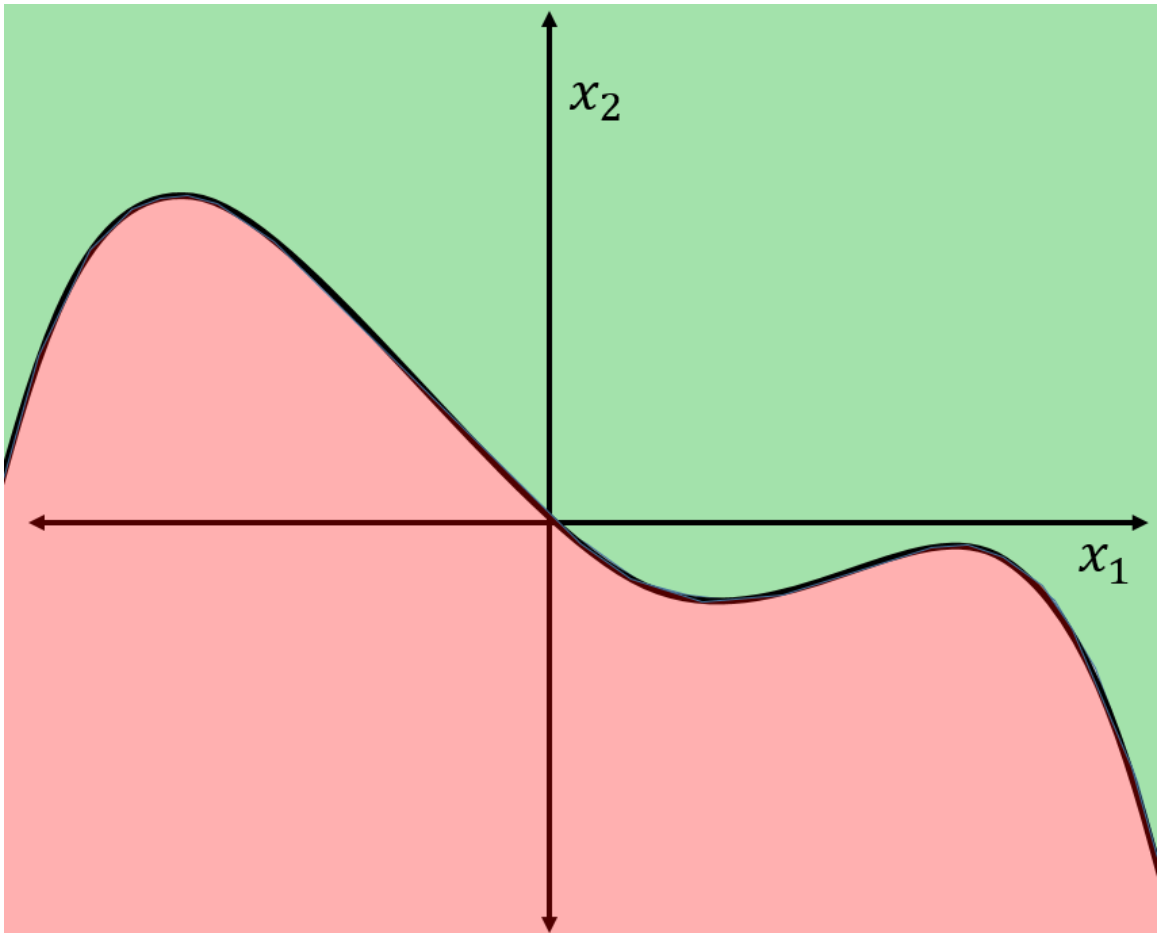

Figure 5-6: An example of a phase plot with regions of the domain separated into possible stable and unstable regions.

Similarly,

$$Q = \{Q_{ij}\} = \{\langle g_i(x;\theta_i) \circ f(x), g_j(x;\theta_j)\rangle\} \tag{5.15}$$

Note that simulation data are used only for finding effective observables and not for obtaining the state transition A matrix. The A matrix is obtained from DE by using the neural net observables as shown in the above equation. Algorithm 2 summarizes this procedure, referred to in latter sections as the SSOG Model, as it joins together two neural network models that are constructed to generate observables specific to the subspaces they are trained on.

Algorithm 2: Training scheme for the prediction models. Notation is explained in Table 5.3.

Input:
G_u: $x_u \sim X_u$; G_s: $x_s \sim X_s$; G_{joint}: $x \sim X$;
F_u: z_{u_k}; F_s : z_{s_k}; F_j: z_{j_k}
Parameters: (weight variables);
Output:
G_u: g_u; G_s: g_s; G_{joint}: g_{joint}
F_u: $\hat{z}_{u_{k+1}}$; F_s: $\hat{z}_{s_{k+1}}$; F_j: $\hat{z}_{j_{k+1}}$
for *number of epochs* **do**
    Sample x_* from X_*
    Generate observable function vector g_*, by inputting x_* through nonlinear layers
    Append state vector to observable vector $z_* = x_* \oplus g_*$
    Estimate augmented output through linear activation $F_*(z_{*_k})$
    Forward pass through MSE Loss function $L(z, \hat{z})$;
    Backward pass: update parameters for observable functions (W_u, W_s);
then
Calculate augmented input g_{u_k} and g_{s_k} for $x_{k_{DE}}$ X_{DE} for all X_{DE}
Append augmented inputs $z_{k_{DE}} = x_{k_{DE}} \oplus g_{u_k} \oplus g_{s_k}$
Calculate augmented output $g_{u_{k+1}}$ and $g_{s_{k+1}}$ for $x_{k+1_{DE}}$ X_{DE} for all X_{DE}
Append augmented outputs $z_{k+1_{DE}} = x_{k+1_{DE}} \oplus g_{u_{k+1}} \oplus g_{s_{k+1}}$
Calculate state transition matrix for linear layer of joint model

Notation	Definition
x	state vector
x_u	state vector belonging to unstable region
x_s	state vector belonging to stable region
G_*	Observable function model trained on the $*$ dataset
F_*	Linear regression model trained on $*$ dataset
x_{*_k}	state vector at initial time step
$x_{*_{k+1}}$	state vector at next time step
W_*	weights for a model (G_* and F_*)
z	lifted state vector
L	Loss function
$*$	Subscript denotes being used for stable (s), unstable (u) regions

Table 5.3: Notation and definitions for variables indicated in different parts of this paper.

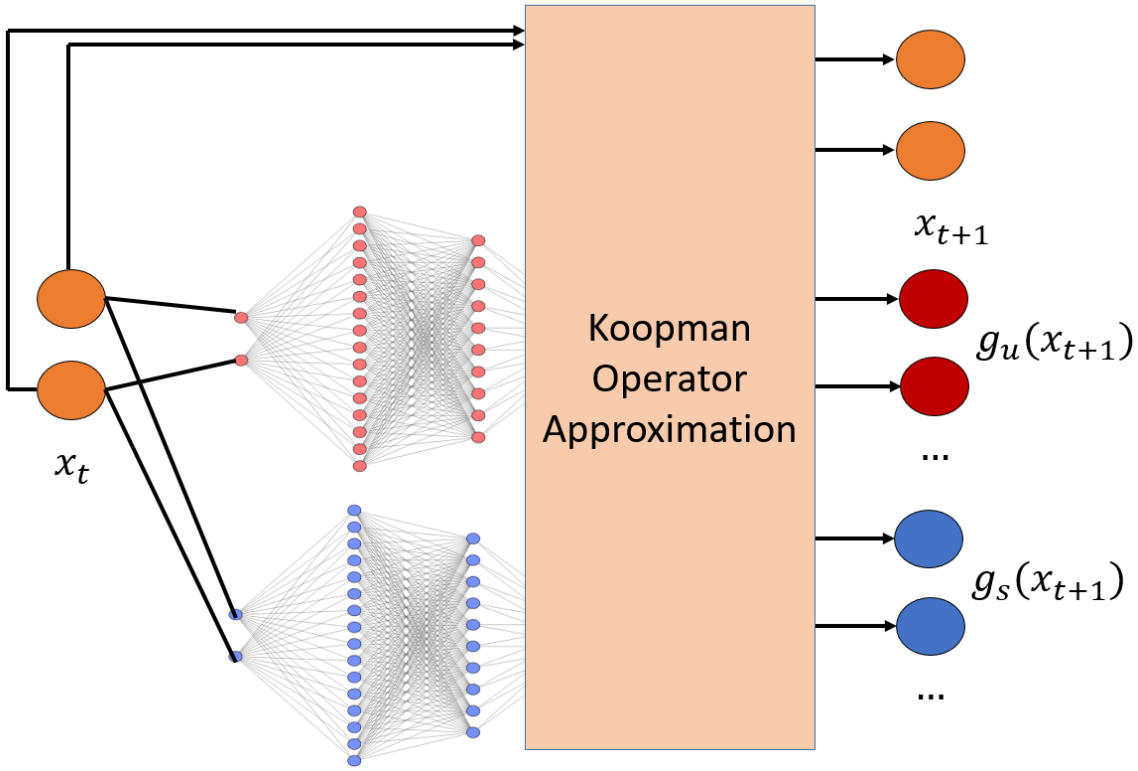

Figure 5-7: Resultant model architecture of the Joint Model described in Algorithm 2.

5.3.1 Results

The computer creating these models has a AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz, and a NVIDIA GeForce GTX 1060 3GB Graphics Card. The models are created using PyTorch. The SSOG models have two hidden layers with ReLU activation units, and the final layer utilizes a linear layer, which represents the Koopman Operator's linear transition matrix. The width of the first hidden layer is 16 units, and the width of the second hidden layer is 10 units. These hidden layers utilize rectified linear activation units. In the case of 40 observables, the SSOG model has 20 observables for both stable and unstable regions. In the case of 80 observables, the division is done similarly. In addition to these observable functions, the original state variables are included as part of the model as well. The loss function for the neural networks is a standard mean squared error loss function of

$$L_{MSE} = \frac{1}{N} \sum_{t} |z_{t+1} - A z_t|^2 \tag{5.16}$$

where z is the lifted state variable, and A is the weights of the final linear layer. The model is trained using an Adam optimizer with a learning rate of $\alpha = 0.01$. Hyperparameters were equivalent between models.

The SSOG model with DE is compared to the SSOG model without the use of DE, where the weights of the linear activation layer is computed from a least squares regression. It is also compared to an Aggregate model which utilizes the same neural network structure but aggregates the dataset together, training a single network that produces twice the number of observable functions and is not subspace specific. A DE version of the Aggregate model is also formulated, recomputing the final layer of the model using DE.

Standard EDMD models are constructed to be of equivalent order to the SSOG and Aggregate without deep learning. The EDMD model's observables are radial basis functions that are uniformly distributed for individual states between minimum and maximum value for that state in the dataset. These radial basis functions are of

the form

$$\psi = e^{-\omega(x_i - x_{i_a})^2} \tag{5.17}$$

where ω is a parameter that was set to equal 1 for all state variables and x_i is the ith state variable, and x_{i_a} is the center of the radial basis function.

We introduce a second order nonlinear system:

$$\dot{x} = -x + x^2 + y^2$$
$$\dot{y} = -y + y^2 + x^2 - x$$

This system has effectively two regions, a stable region and an unstable region, visualized with a collection of trajectories in Fig. 5-8.

The observables generated from training this model on the stable dataset is shown in Fig. 5-9, while the observable functions corresponding to the unstable dataset is shown in Fig. 5-10. When aggregating the datasets together, the observable functions are shown in Fig. 5-11.

The pole plots corresponding to the Aggregate model and the SSOG model are shown in Figs. 5-12, and 5-13. An additional pole plot is provided showing the difference between the models after the use of Direct Encoding in Fig. 5-14.

Prediction Error

The prediction error for this system is calculated separately for stable and unstable region time series trajectories with a set of test data consisting of initial conditions not included for either DE nor the aggregate trajectory dataset. The equation used for prediction error is sum of squared error for the state variables, not including observables of the system. That is

$$E_{sse} = \sum_{i=0}^{N} (x_i - \hat{x}_i)^2 \tag{5.18}$$

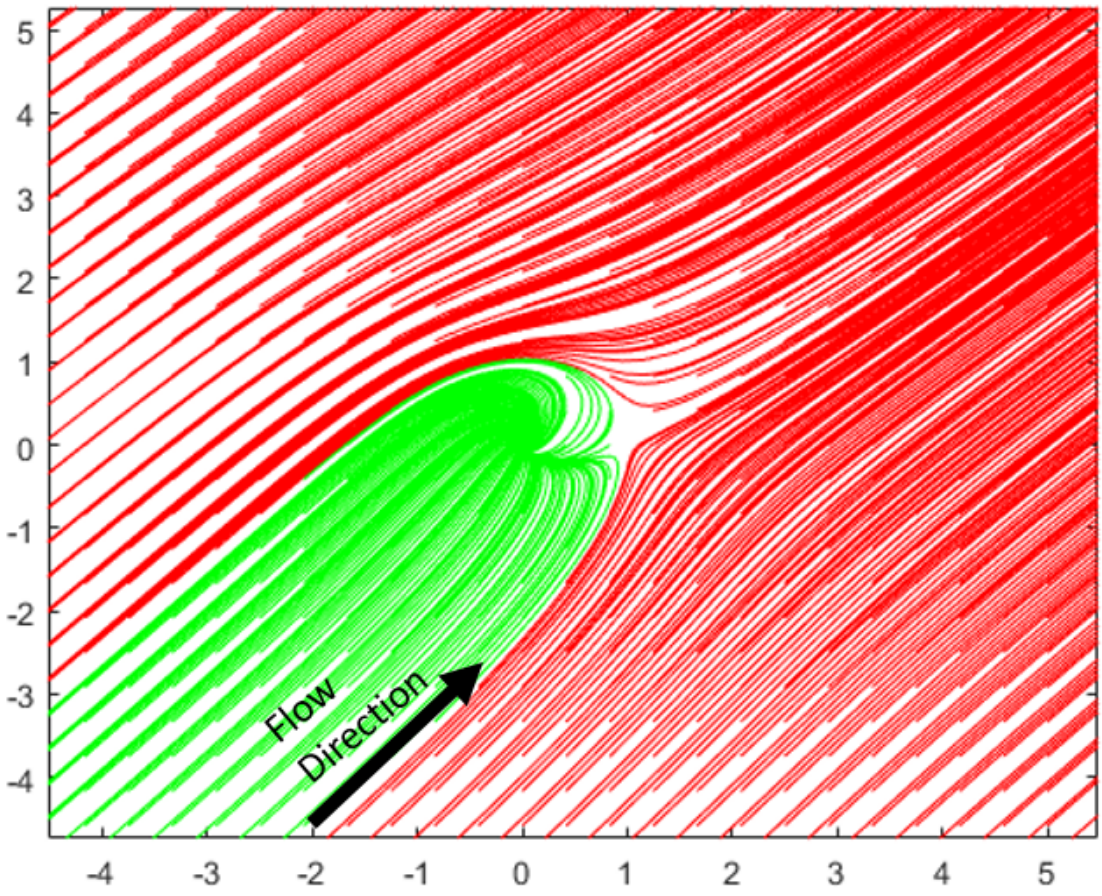

Figure 5-8: Phase Plot visualization of initial conditions that yield stable versus unstable trajectories. The green trajectories symbolize the stable region, and the red trajectories symbolize the unstable region.

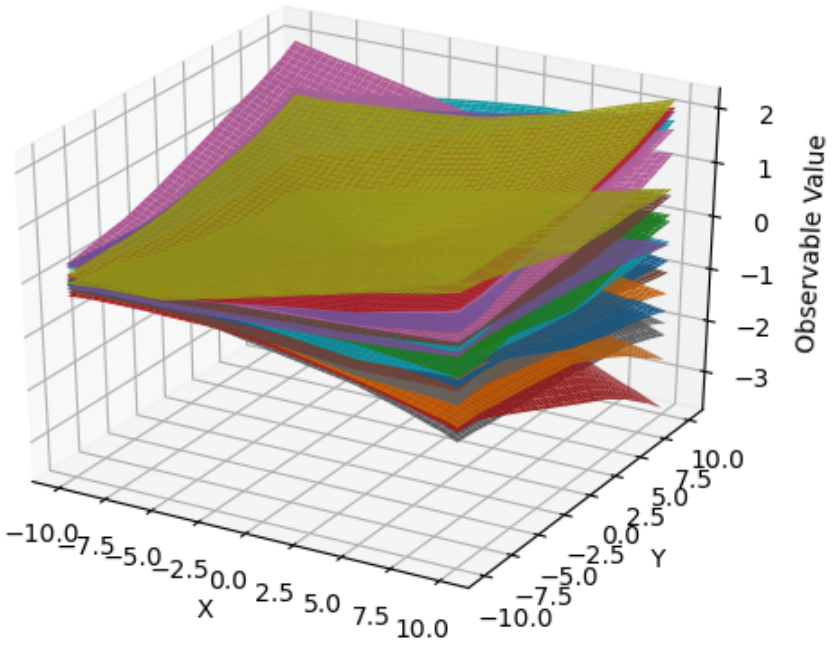

Figure 5-9: Observable functions learned from training on a stable dataset.

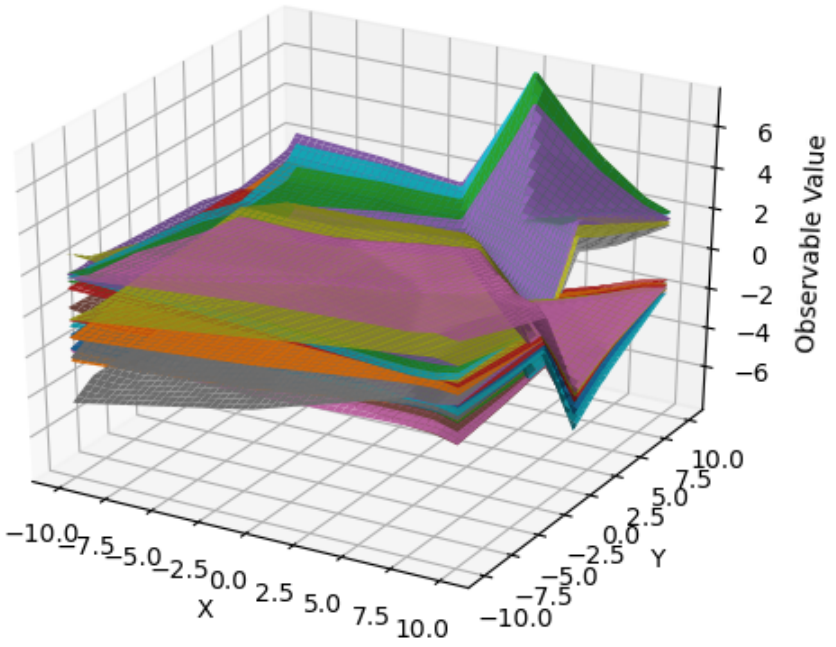

Figure 5-10: Observable functions learned from training on a unstable dataset.

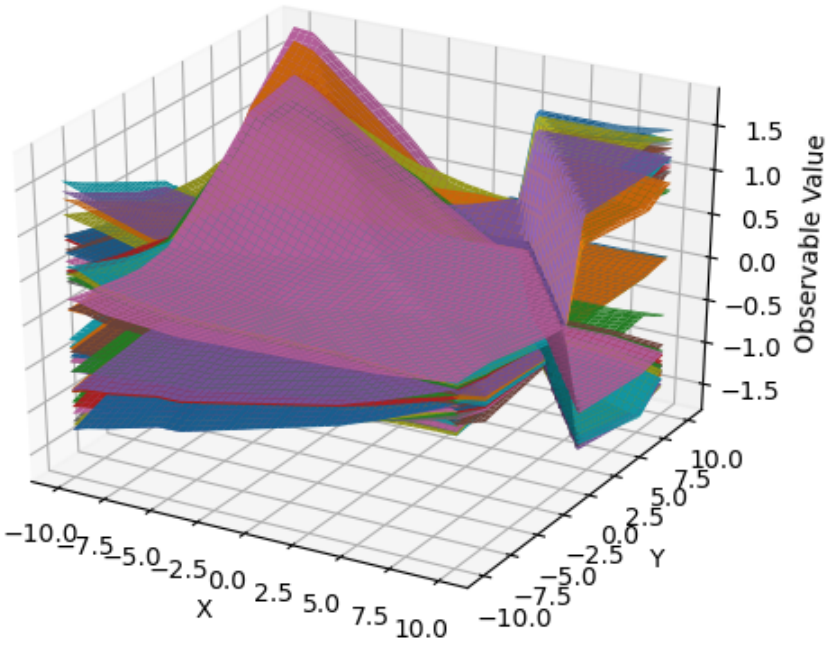

Figure 5-11: Observable functions learned from training on the aggregated dataset.

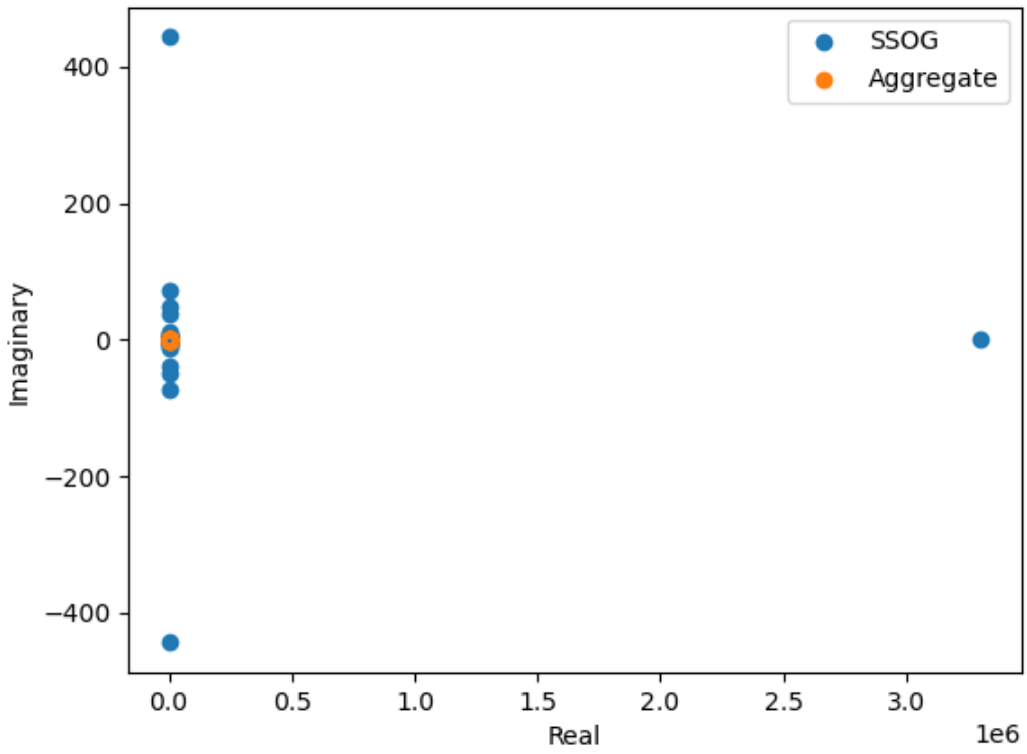

Figure 5-12: Pole plot comparing the eigenvalues of the model trained on the aggregated dataset and the eigenvalues of the SSOG model.

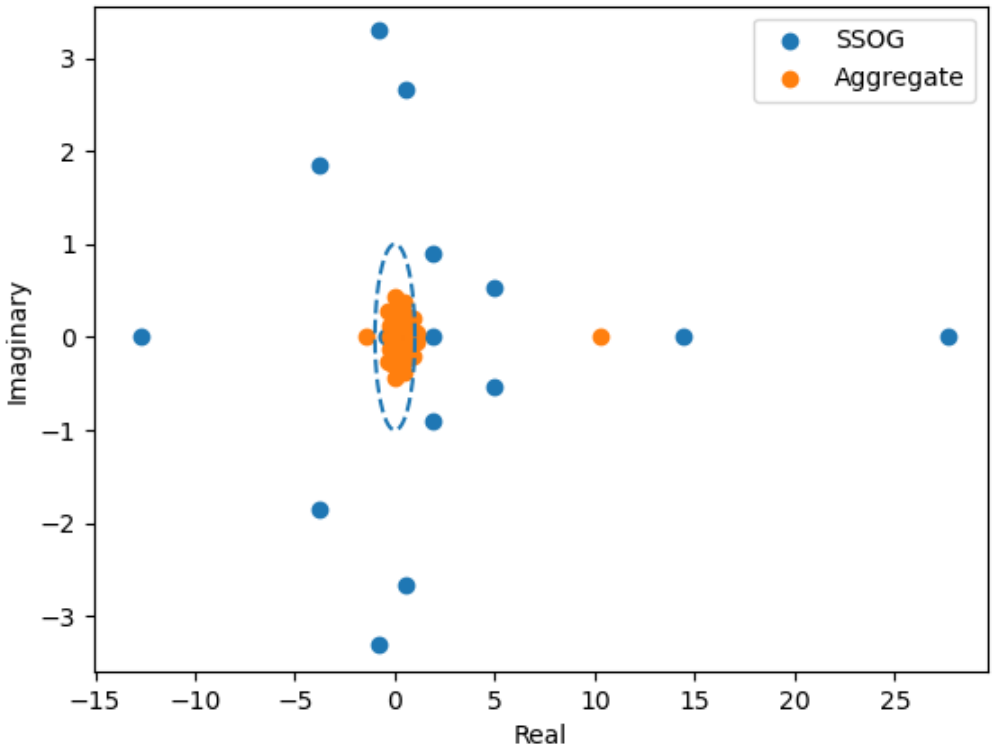

Figure 5-13: A zoomed in perspective of the pole plot comparing the eigenvalues of the model trained on the aggregated dataset and the eigenvalues of the SSOG model. The dashed ellipse represents the unit circle, indicating the region for which eigenvalues belong to stable eigenvectors.

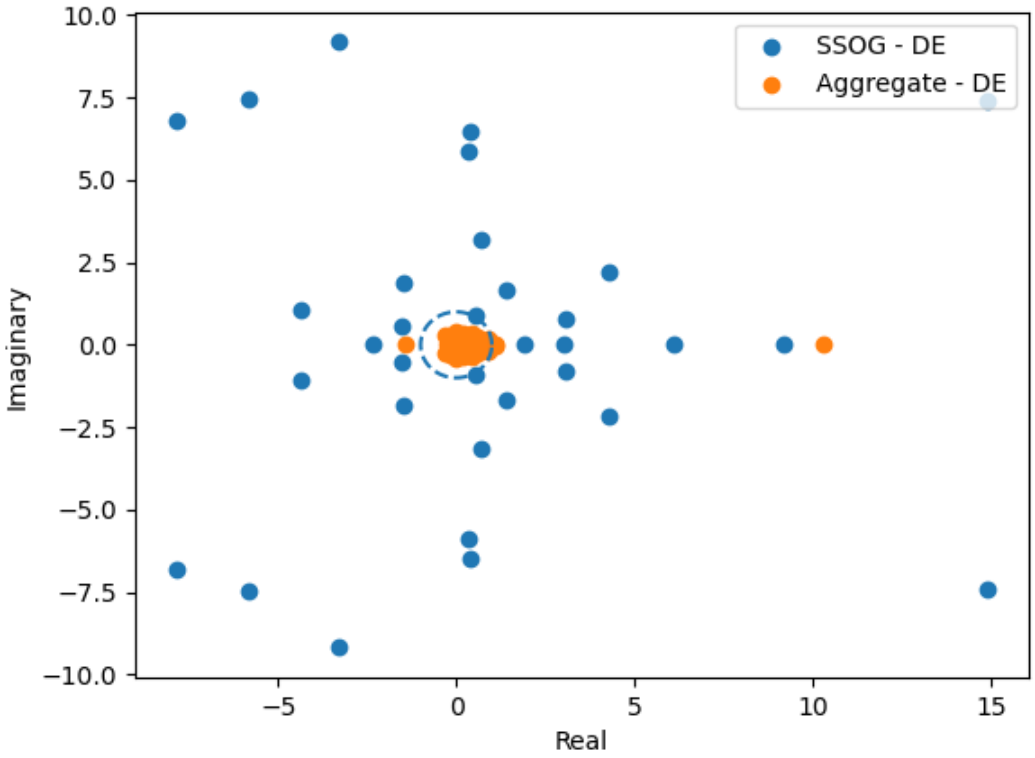

Figure 5-14: A zoomed in perspective of the pole plot comparing the eigenvalues of the model trained on the aggregated dataset and the eigenvalues of the SSOG model with the usage of Direct Encoding. The dashed ellipse represents the unit circle, indicating the region for which eigenvalues belong to stable eigenvectors.

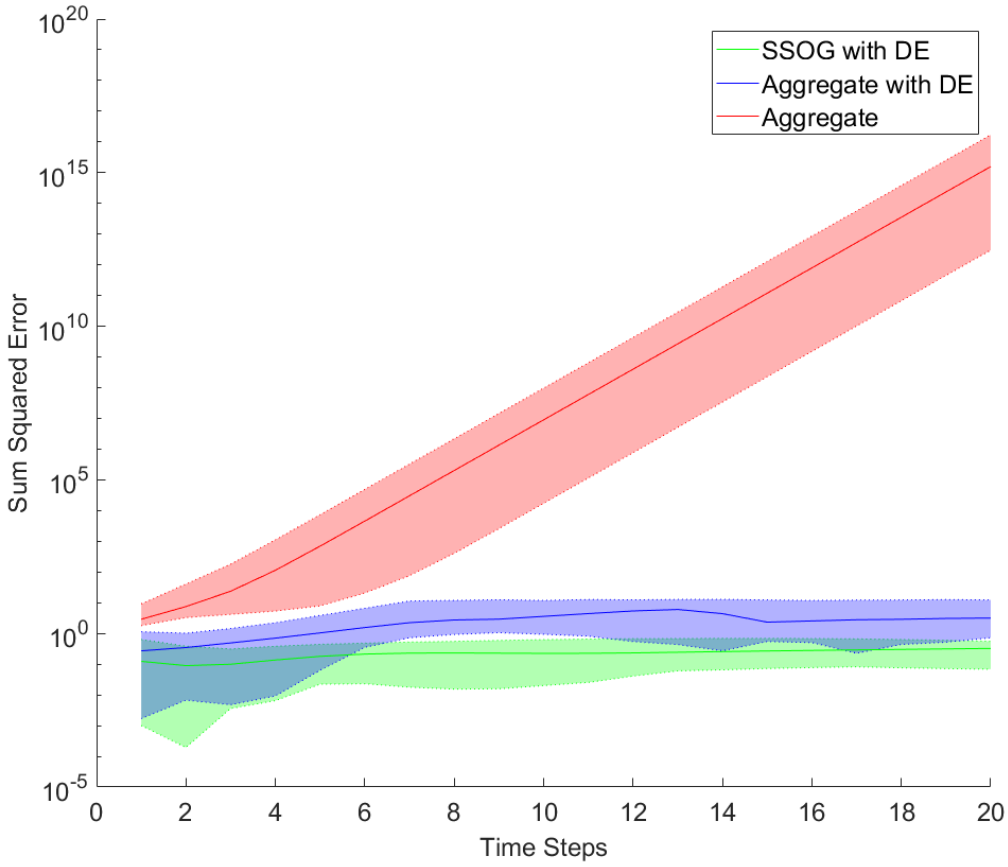

Figure 5-15: Prediction error for 100 trajectories beginning at initial conditions of a test set within the stable subspace. The shaded region represents the variation between minimum and maximum SSE at the given time step for the listed model. The solid lines represent the average sum of squared error. Comparison is between the SSOG with Direct Encoding, and Aggregate Model with and without DE. The models plotted utilize 40 observables in addition to the two state variables.

where i is the ith state variable for a given time step. The estimated state variable, $\hat{x}$, is the prediction from the model, where the ground truth is denoted as x. This prediction error is not of the total lifted state, but of the state variables belonging to the original nonlinear system which are shared between all models.

The prediction error is visualized in the set of figures, Fig. 5-15. The two figures compare the SSOG with DE and other learned models. Further comparisons using this test set are outlined in Table 5.4. The best performing result of the labeled order for each subspace for each set of models is in bold.

Table 5.4: Average SSE prediction error after 1 and 10 time steps for trajectories in set of test data.

Method	Stable	Unstable
40 Observables, 1 Time Step		
SSOG	2.84	6.15
SSOG with DE	**0.12**	**0.26**
Aggregate	2.90	5.23
Aggregate with DE	0.27	1.44
EDMD	3.79×10^3	2.06×10^3
80 Observables, 1 Time Step		
SSOG	4.16	4.80×10^5
SSOG with DE	**0.11**	**0.36**
Aggregate	2.20	2.83
Aggregate with DE	0.60	1.12
EDMD	1.28×10^7	7.71×10^6
40 Observables, 10 Time Steps		
SSOG	2.50×10^{36}	6.8×10^{131}
SSOG with DE	**0.22**	6.8×10^{131}
Aggregate	9.00×10^6	6.8×10^{131}
Aggregate with DE	3.60	6.8×10^{131}
EDMD	2.15×10^{35}	6.8×10^{131}
80 Observables, 10 Time Steps		
SSOG	5.71×10^{10}	6.8×10^{131}
SSOG with DE	**0.18**	6.8×10^{131}
Aggregate	1.06×10^8	6.8×10^{131}
Aggregate with DE	11.63	6.8×10^{131}
EDMD	2.10×10^{73}	6.8×10^{131}

Subspace Boundary Formation

Finding the boundary between stable and unstable regions is important for analyzing nonlinear dynamics. Here we compare each modeling method in terms of the accuracy in finding the boundary. Consider the following quotient:

$$\xi = \frac{||z_u||}{||z||} \tag{5.19}$$

where

$$z_u = W_u^T z \tag{5.20}$$

If z is in a stable region and the matrix W_u spanning the unstable region is accurate, then the quotient ξ must be 0. On the other hand, if W_u is inaccurate, then some fraction of the component will enter the stable region with non-zero ξ. Fig 5-8 shows the plots of this quotient indicating how accurately the true boundary is recreated.

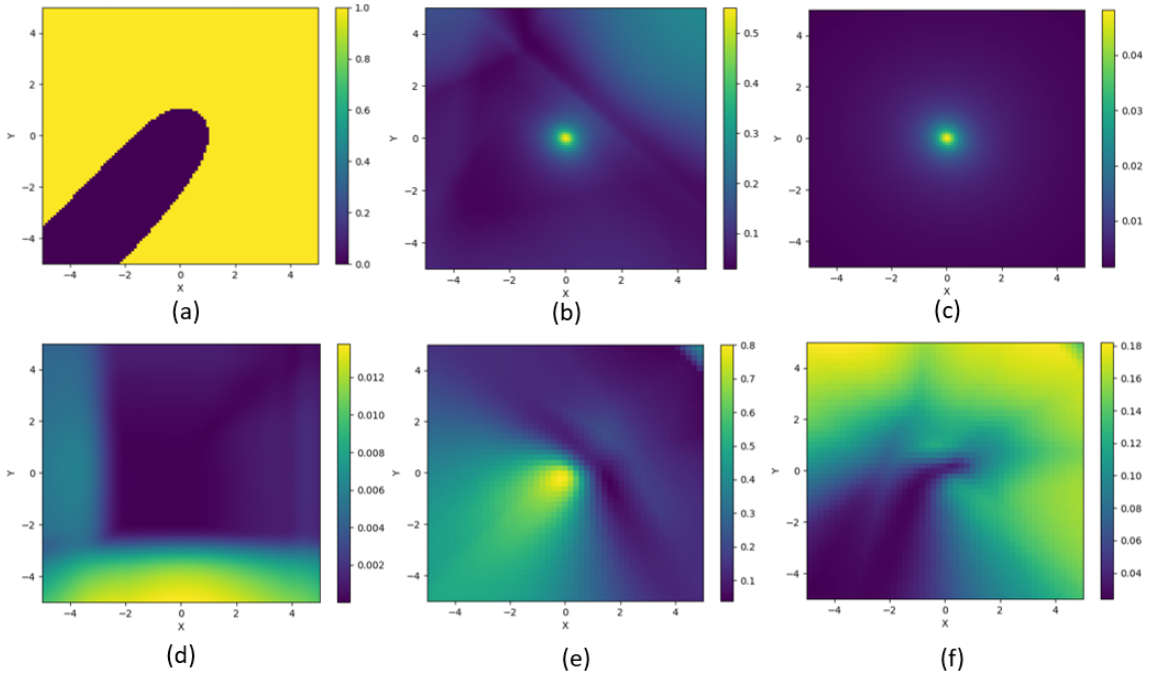

Figure 5-16: Instability quotient plots for models of the system. Top: (a) Ground Truth assuming complete separation, (b) Aggregate Model, (c) Aggregate Model with Direct Encoding. Bottom: (d) EDMD, (e) SSOG Model, (f) SSOG Model with Direct Encoding. By comparing all other plots to (a), it is clear that in terms of shape (f) most accurately resembles the ground truth.

5.3.2 Discussion

The first key result found from Table 5.4 is that using SSOG in combination with DE yields the highest accuracy. Secondly, the DE method is shown to drastically lower the prediction error for both Aggregate and SSOG models by several orders of magnitude. This result matches the expectation that DE removes biases from over- or undersampling of a region given its formulation that involves calculating inner products over a domain. Furthermore, the DE method is demonstrated to have a significant impact even with increases in order of the system. Notably however, none

of the models are capable of predicting trajectories in the unstable subspace.

When examining Figs. 5-9, 5-10, and 5-11 that regard the observable functions themselves, it is clear that training with different datasets yields different observable functions. When examining Figs. 5-12, 5-13, and 5-14, the difference in accuracy for the SSOG models versus the Aggregate models appears to be due to the fact that the resulting linear models are inherently different from each other from a modal perspective. Without the use of Direct Encoding, both models are composed of far more unstable modes. In the case of the Aggregate models, the use of Direct Encoding significantly changes the shape of the plot, but the main locations of the eigenvalues remain the same. However, for SSOG, the use of DE drastically changes the arrangement of the poles for the linear model, bringing them far closer to the unit circle. This change provides a mathematical explanation for the accuracy of the SSOG model with DE for longer trajectories.

Because the prediction models used are finite order approximations of the Koopman operator, we expect inaccuracies to arise when using these models to locate boundaries between regions. For that reason, the instability quotient is introduced. Assuming complete separation of dynamic modes, the ratio of unstable projection to state vector (instability quotient) should be 0 for the stable region, and 1 for the unstable region, as shown in plot (a) of Fig. 5-16. However, due to the approximation of the system as a linear system, this discrete switch becomes blurred, depending on the accuracy of the model which is shown in the other plots. The SSOG Model with DE demonstrates that the instability quotient increases significantly in the unstable region. This result does not occur for the other models, further indicating that the observables learned for the SSOG Model are effective for their specific subspaces. The DE method is also key in this result as it demonstrates a drastic change in the dynamic model.

5.4 Conclusion

In this chapter, presented two significant results. The first was incorporation of DDE to Deep Koopman, i.e. neural network based methods for construction of the Koopman Operator, for improving prediction accuracy. The second significant result, which corresponds to the third contribution of the thesis, was the formulation of the subspace specific observable generation (SSOG) method for learning an efficient set of observable functions to lift the state space of nonlinear dynamic systems. In combination with an existing method, Direct Encoding (DE), SSOG was shown to improve accuracy given certain conditions. Separately, SSOG with DE demonstrates a capacity to be used as an analysis tool for finding borders between subspaces based on analytical foundations. The work has a clear direction for improvement; the current network structure and training does not enable the subspaces to be fully separated as the observable functions learned are not zero outside of their respective regions. This would be a notable direction to explore in the future to further improve results.

Chapter 6

Conclusion and Future Work

In this work, we explored the construction of the Koopman Operator for nonlinear systems. With prior work in mind, we discussed the next class of systems, systems with segmented dynamics, that stand to benefit the most from linearizing through the construction of the Koopman Operator. We demonstrate the success in modeling these systems with linear models and applying Model Predictive Control (MPC). The trajectory generated from MPC demonstrates some degree of intelligent trajectory planning. However, it is shown in Chapter 4 that the use of Least Squares Estimation (LSE) to solve for the Koopman Operator has unintended effects depending on the distribution of the datasets used to learn these models. By adopting a data-driven version of Direct Encoding, dubbed Data-Driven Encoding, models learned with this numerical integration-based method outperform the LSE based models. This difference in performance is driven by the underlying mechanics in the formulations. Further, by utilizing Direct Encoding and Data-Driven Encoding in combination with Deep Learning of the Koopman Operator, the types of nonlinear system behaviors that can be predicted with the resulting linear models widens, allowing for application to unstable systems, and also lowers the overall order required to represent these models.

This thesis touches upon each avenue for extending the use of the Koopman Operator, from pushing the frontier of how the theory can be applied to nonlinear systems of different classes, but also for more accurately formulating the operator both based

on observable functions and the linear transition matrix. However, there still exist several directions to continue. Through the creation of Data-Driven Encoding, a significant number of problems that were problematic to form linear models for may now be tractable. An example of such a system that was puzzling and difficult to linearize are walking robots, both passive and actuated, as they exist in the realm of systems defined as hybrid systems. With regards to the formulation of the linear transition matrix, the use of Data-Driven Encoding has demonstrated increased accuracy, but through the change in formulation has lost the capability of being easily updated. The use of EDMD allowed for linear updates in a recursive fashion with simple matrix math; this updating method is not as easily computed for Data-Driven Encoding as it requires insertion into a graph. Producing such a method that would allow for real-time updates would be incredibly beneficial. Similarly, the new method does not have a simple formulation that is mathematically rigorous for applications of control and currently exists solely for autonomous systems, and the creation of such a formulation would drastically increase the opportunities for which the method can be applied.

Another direction that remains puzzling is the formation of unstable modes for stable systems with specific selections of observable functions. Though prior work has discussed the issue, it remains a problem that is only ameliorated by algorithms that stray away from the core theory of the Koopman Operator. Data-Driven Encoding does not resolve this issue, though preliminary work has found that the instability forms differently when utilizing that method for formulation of the linear matrix as opposed to EDMD.

Perhaps most important, though only discussed for a single chapter, is the possible application of the Koopman Operator to Deep Learning. Though modern problems tackled by Deep Learning vary widely, many of which constitute modeling functions that are of similar formulation as the nonlinear dynamic systems for which the formulation of the Koopman Operator has been applied on. As the bare-minimum to apply the theory is a self-map, a nonlinear one-to-one transition between one state and another, the number of problems that remain to be solved is vast and innumer-

able. But, similar to how observables are chosen for construction of the Koopman Operator, we must choose wisely and choose what is most important for our goals.

Bibliography

[1] Ian Abraham and Todd D. Murphey. Active learning of dynamics for data-driven control using koopman operators. *CoRR*, abs/1906.05194, 2019.

[2] Mustaffa Alfatlawi and Vaibhav Srivastava. An incremental approach to online dynamic mode decomposition for time-varying systems with applications to eeg data modeling. *arXiv preprint arXiv:1908.01047*, 2019.

[3] Berk Altın, Pegah Ojaghi, and Ricardo G Sanfelice. A model predictive control framework for hybrid dynamical systems. *IFAC-PapersOnLine*, 51(20):128–133, 2018.

[4] Berk Altın, Pegah Ojaghi, and Ricardo G. Sanfelice. A model predictive control framework for hybrid dynamical systems. *IFAC-PapersOnLine*, 51(20):128–133, 2018. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

[5] Harry Asada. Global, unified representation of heterogenous robot dynamics using composition operators. *arXiv preprint arXiv:2208.00175*, 2022.

[6] Petar Bevanda, Max Beier, Sebastian Kerz, Armin Lederer, Stefan Sosnowski, and Sandra Hirche. Diffeomorphically learning stable koopman operators. *IEEE Control Systems Letters*, 6:3427–3432, 2022.

[7] Daniel Bruder, Xun Fu, R. Brent Gillespie, C. David Remy, and Ram Vasudevan. Data-driven control of soft robots using koopman operator theory. *IEEE Transactions on Robotics*, 37(3):948–961, 2021.

[8] Steven L Brunton, Marko Budišić, Eurika Kaiser, and J Nathan Kutz. Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*, 2021.

[9] Eduardo F Camacho, Daniel R Ramírez, Daniel Limón, D Muñoz De La Peña, and Teodoro Alamo. Model predictive control techniques for hybrid systems. *Annual reviews in control*, 34(1):21–31, 2010.

[10] Lee-Huang Chen, Kyunam Kim, Ellande Tang, Kevin Li, Richard House, Edward Liu Zhu, Kimberley Fountain, Alice M. Agogino, Adrian Agogino, Vytas Sunspiral, and Erik Jung. Soft Spherical Tensegrity Robot Design Using Rod-Centered Actuation and Control. *Journal of Mechanisms and Robotics*, 9(2), 03 2017. 025001.

[11] Siu-Wing Cheng, Tamal Krishna Dey, Jonathan Shewchuk, and Sartaj Sahni. *Delaunay mesh generation*. CRC Press Boca Raton, 2013.

[12] Vít Cibulka, Tomáš Haniš, Milan Korda, and Martin Hromčík. Model predictive control of a vehicle using koopman operator. *IFAC-PapersOnLine*, 53(2):4228–4233, 2020. 21st IFAC World Congress.

[13] Suddhasattwa Das. Lie group valued koopman eigenfunctions. *arXiv preprint arXiv:1808.09590*, 2018.

[14] Philip J Davis and Philip Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

[15] Fletcher Fan, Bowen Yi, David Rye, Guodong Shi, and Ian R Manchester. Learning stable koopman embeddings. In *2022 American Control Conference (ACC)*, pages 2742–2747. IEEE, 2022.

[16] David A Freedman. *Statistical models: theory and practice*. Cambridge University Press, 2009.

[17] Clément Gosselin, Philippe Cardou, Tobias Bruckmann, and Andreas Pott. *Cable-Driven Parallel Robots: Proceedings of the Third International Conference on Cable-Driven Parallel Robots*, volume 53 of *Mechanisms and Machine Science*. Springer International Publishing AG, Cham, 2017.

[18] Minghao Han, Jacob Euler-Rolle, and Robert K Katzschmann. Desko: Stability-assured robust control with a deep stochastic koopman operator. In *International Conference on Learning Representations*, 2021.

[19] Yiqiang Han, Wenjian Hao, and Umesh Vaidya. Deep learning of koopman representation for control. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1890–1895. IEEE, 2020.

[20] Rachel Hoffman and H. Harry Asada. Precision assembly of heavy objects suspended with multiple cables from a crane. *IEEE Robotics and Automation Letters*, 5(4):6876–6883, 2020.

[21] Yusuke Igarashi, Masaki Yamakita, Jerry Ng, and Harry Asada. Mpc performance for nonlinear systems using several linearization models. *2020 American Control Conference*, 35:718–730, July 2020.

[22] Mason Kamb, Eurika Kaiser, Steven L Brunton, and J Nathan Kutz. Time-delay observables for koopman: Theory and applications. *SIAM Journal on Applied Dynamical Systems*, 19(2):886–917, 2020.

[23] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[24] M. Korayem, H. Tourajizadeh, and M. Bamdad. Dynamic load carrying capacity of flexible cable suspended robot: Robust feedback linearization control approach. *Journal of Intelligent & Robotic Systems*, 60:341–363, 2010.

[25] Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *arXiv e-prints*, page arXiv:1611.03537, Nov 2016.

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

[27] Milan Korda and Igor Mezić. Optimal construction of koopman eigenfunctions for prediction and control. *IEEE Transactions on Automatic Control*, 65(12):5114–5129, 2020.

[28] Sangheon Lee and Hungsun Son. Antisway control of a multirotor with cable-suspended payload. *IEEE Transactions on Control Systems Technology*, pages 1–9, 2020.

[29] Taeyoung Lee. Geometric control of quadrotor uavs transporting a cable-suspended rigid body. *IEEE Transactions on Control Systems Technology*, 26(1):255–264, 2018.

[30] Changquing Li and Christopher D. Rahn. Design of Continuous Backbone, Cable-Driven Robots . *Journal of Mechanical Design*, 124(2):265–271, 05 2002.

[31] Jonqlan Lin and Guan-Ting Liao. A modularized cable-suspended robot: Implementation and oscillation suppression control. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 230(9):1030–1043, 2016.

[32] Lennart Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

[33] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1), Nov 2018.

[34] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

[35] Giorgos Mamakoukas, Ian Abraham, and Todd D Murphey. Learning stable models for prediction and control. *IEEE Trans Robot*, 2020.

[36] G Manjunath and A de Clercq. Universal set of observables for the koopman operator through causal embedding, 2021.

[37] Philippe Manon, Claire Valentin-Roubinet, and Gérard Gilles. Optimal control of hybrid dynamical systems: application in process engineering. *Control Engineering Practice*, 10(2):133–149, 2002.

[38] Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *IEEE Transactions on Automatic Control*, 65(6):2550–2565, 2019.

[39] Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[40] Igor Mezić. Spectrum of the koopman operator, spectral expansions in functional spaces, and state-space geometry. *Journal of Nonlinear Science*, 30(5):2091–2145, 2020.

[41] Kamil Nar, Yuan Xue, and Andrew M Dai. Learning unstable dynamical systems with time-weighted logarithmic loss. *arXiv preprint arXiv:2007.05189*, 2020.

[42] Jerry Ng and H Harry Asada. Model predictive control and transfer learning of hybrid systems using lifting linearization applied to cable suspension systems. *IEEE Robotics and Automation Letters*, 7(2):682–689, 2021.

[43] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[44] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[45] Peter J. Schmid. Dynamic mode decomposition and its variants. *Annual Review of Fluid Mechanics*, 54(1):225–254, 2022.

[46] Nicholas S Selby, Filippos E Sotiropoulos, and H Harry Asada. Physics-based causal lifting linearization of nonlinear control systems underpinned by the koopman operator. *arXiv preprint arXiv:2108.10980*, 2021.

[47] Simone Servadio, David Arnas, and Richard Linares. A koopman operator tutorial with othogonal polynomials. *arXiv preprint arXiv:2111.07485*, 2021.

[48] Filippos E Sotiropoulos and H Harry Asada. Dynamic modeling of bucket-soil interactions using koopman-dfl lifting linearization for model predictive contouring control of autonomous excavators. *IEEE Robotics and Automation Letters*, 7(1):151–158, 2021.

[49] Robert Vrabel. Design of the state feedback-based feed-forward controller asymptotically stabilizing the overhead crane at the desired end position. *Journal of Applied Nonlinear Dynamics*, 03 2019.

[50] 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.

[51] Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, pages 4832–4839. IEEE, 2019.

[52] Hao Zhang, Clarence W Rowley, Eric A Deem, and Louis N Cattafesta. Online dynamic mode decomposition for time-varying systems. *SIAM Journal on Applied Dynamical Systems*, 18(3):1586–1609, 2019.

[53] Shaoxiong Zhang, Yunhong Wang, and Annan Li. Cross-view gait recognition with deep universal linear embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9095–9104, 2021.