Leveraging Neural Networks and the Koopman Operator for Controlled Dynamical Systems and Linear Model Predictive Control

by

Sriraj Meenavilli

B.Sc., University of Regina, 2019

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Sriraj Meenavilli**
Degree:	**Master of Science**
Thesis title:	**Leveraging Neural Networks and the Koopman Operator for Controlled Dynamical Systems and Linear Model Predictive Control**
Committee:	**Chair:** Angel Chang Assistant Professor, Computing Science
	Mo Chen Supervisor Assistant Professor, Computing Science
	Sharan Vaswani Committee Member Assistant Professor, Computing Science
	Max Libbrecht Examiner Associate Professor, Computing Science

Abstract

In this thesis, we present a novel network architecture using an encoder-decoder neural network, inspired by Lusch et al. [15], to directly obtain the eigenfunctions of the Koopman operator, which globally linearize the dynamics of nonlinear systems. By extending this approach to account for control inputs, we enable the application of well-established control synthesis techniques for obtaining optimal policies, while ensuring that control constraints remain convex for globally optimal solutions. This method overcomes the limitations of traditional Koopman operator approximations and provides improved prediction and control performance. We demonstrate the efficacy of our approach on a range of simulated controlled dynamical systems and show its potential for real-world applications in fields like robotics, and process control.

Keywords: Koopman operator; Deep Neural Networks; Linear Model Predictive Control (MPC); System identification; Controlled dynamical systems

Dedication

To my esteemed supervisor, Dr. Mo Chen, for the invaluable guidance, patience, and expert knowledge that have been the bedrock of this research. Your mentorship has not only shaped this work but also my journey as a scholar and a person.
To my Mother, Father, and Brother, whose unyielding support and understanding provided me with the strength to persevere through the challenges, including the unprecedented times brought upon by COVID-19. This accomplishment stands as a testament to all of you, who have contributed to my growth and to the completion of this project.
Thank you all, I would have never gotten this far without you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Controlled dynamical systems play a crucial role in numerous applications, including robotics, aerospace, and process control [13]. The inherent complexity of these systems often arises from their nonlinear nature, which complicates system modeling, prediction, and control [4]. Consequently, researchers have sought methods to linearize nonlinear systems to leverage the benefits of linear control techniques and simplify optimization problems related to optimal control [10].

The Koopman operator, introduced by Bernard Koopman in 1931 [11], has gained traction in recent years. Koopman's idea is to globally linearlize nonlinear systems by representing their dynamics in a linear form within an infinite-dimensional function space [4]. Despite its potential, traditional methods for approximating the Koopman embedding function and Koopman operator, such as Dynamic Mode Decomposition (DMD)[22], have encountered various challenges, including difficulties in capturing the intricate dynamics of complex systems, leading to suboptimal predictions [24].

With the advent of deep learning techniques, neural networks have exhibited remarkable success in modeling intricate systems across diverse domains [23]. Encoder-decoder architectures, in particular, have demonstrated potential in addressing the limitations of conventional Koopman-based approaches by learning the eigenfunctions responsible for global linearization directly [15]. These networks offer the possibility of more accurate and robust predictions, as well as improved control performance [6].

1.2 Goal

The primary objective of this research is to develop an effective data-driven method for finding linear dynamical models for controlled nonlinear systems by extending the work of Lusch et al. [15], who used a deep neural network to directly learn the eigenfunctions of the Koopman operator, circumventing the need for prior knowledge of system dynamics. We aim to further improve the methodology by incorporating control inputs within the modified encoder-decoder architecture to obtain a linear representation of controlled dynamical systems while preserving the convexity of the optimization problem associated with optimal control.

1.3 Methodology

The proposed methodology involves an encoder-decoder network that maps the system state to a latent space in which the system dynamics are linear. By adjusting the architecture of the neural network to include control inputs, we obtain a linear representation of a controlled dynamical system [13, 6]. We demonstrate the effectiveness of our approach through two numerical simulations involving the 4-D car system, Pole-cart, and a complex industrial system involving a gas pipeline.

Chapter 2

Background

2.1 Koopman Operator

The Koopman operator, first introduced by Bernard Koopman in 1931 [11], is a linear operator that provides a powerful framework to study nonlinear dynamical systems. The idea behind the Koopman operator is to transform a nonlinear system into an infinite-dimensional linear system using an eigenfunction. This in turn allows for the application of linear analysis techniques to understand and control the underlying nonlinear dynamics in a discrete-time setting [17]. Consider a discrete-time nonlinear dynamical system described by:

$$x_{k+1} = f(x_k), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the state vector and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a smooth vector field. The Koopman operator, denoted by $\mathcal{K}$, is defined to act on a function $g : \mathbb{R}^n \to \mathbb{R}^m$, as follows:

$$(\mathcal{K}g)(x_k) = g(f(x_k)) = g(x_{k+1}). \tag{2.2}$$

In other words, the Koopman operator advances the function g one step forward along the system's trajectory. The key insight behind the Koopman operator is that, although the system's dynamics are nonlinear, the operator itself is linear. This linearity enables the use of powerful linear analysis techniques, such as eigenfunction analysis, to study the system's behavior [10]. $(\mathcal{K}g)(x)$ represents the action of the Koopman operator on the function g. The Koopman operator $\mathcal{K}$ is a linear operator, but it acts on the space of observables (functions like g) rather than directly on the state space. It means that if you first apply the nonlinear system dynamics f to the state x, and then apply the function g to the result, this composition is the same as what the Koopman operator does to the function g.

To represent the nonlinear dynamics in a linear framework, the Koopman operator relies on the concept of eigenfunctions. An eigenfunction ψ of the Koopman operator is a function that satisfies the following equation:

$$\mathcal{K}\psi(x) = \psi(f(x)) = \lambda\psi(x), \tag{2.3}$$

where λ is the corresponding eigenvalue. By finding a suitable set of eigenfunctions, one can construct a global linearization of the nonlinear dynamics in the form of the eigendecomposition of the linear map. Based on Eq. (2.2), a set of eigenfunctions $\psi(x)$ would form the eigenbasis for the codomain of $g(\cdot)$; thus in practice, and as we will do in this thesis, the linear map $\mathcal{K}$ can be represented in Jordan canonical form without loss of generality. However, identifying these eigenfunctions analytically is often infeasible for most systems, which has led to the development of various numerical approximation methods, such as Dynamic Mode Decomposition (DMD) [22] and Extended Dynamic Mode Decomposition (eDMD) [24]. In the next section we will talk about how a network architecture is fomulated to satify the framework of the Koopman Operator.

2.2 Learning Lifting Functions

Eigenfunctions play a crucial role in the study of nonlinear dynamical systems through the Koopman operator framework. By finding a suitable set of eigenfunctions, one can construct a global linearization of the nonlinear dynamics. However, identifying these eigenfunctions analytically is often infeasible for most systems, which has led to the development of various numerical approximation methods.

Dynamic Mode Decomposition (DMD) is a popular method to approximate the eigenfunctions of the Koopman operator [22]. DMD operates by fitting a linear combination of exponentials to a set of time-ordered data snapshots. The DMD modes, which are the eigenvectors of the matrix formed from the data snapshots, serve as approximations to the Koopman operator's eigenfunctions. However, DMD relies on the assumption that the underlying dynamics are linear and can result in poor approximations for strongly nonlinear systems.

Extended Dynamic Mode Decomposition (eDMD) extends DMD by including a dictionary of nonlinear basis functions [24]. The basis functions are used to project the state space onto a higher-dimensional space, where the dynamics are more likely to be linear, and then DMD is applied to this higher-dimensional space. The resulting eDMD modes are better approximations of the Koopman eigenfunctions compared to the DMD modes, but the choice of basis functions requires some prior knowledge of the system's dynamics.

Recently, deep learning techniques have been employed to learn the eigenfunctions of the Koopman operator directly from data, offering a more flexible and data-driven approach [15]. By leveraging deep neural networks, researchers can bypass the need for prior knowledge of system dynamics and obtain improved approximations of the Koopman operator. In this approach, an encoder-decoder architecture is used to map the system's state to a latent space in which the dynamics are linear, as highlighted by the central component of Figure

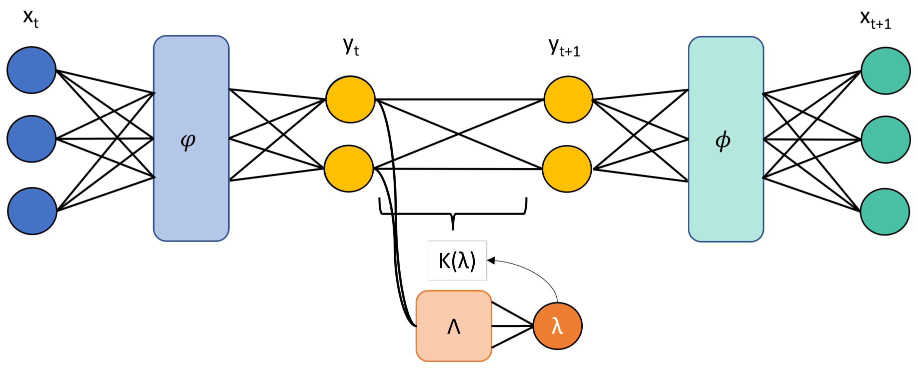

Figure 2.1: Network architecture to identify spectral properties of uncontrolled dynamical systems, highlighting the central role of the linear transformation in the latent space for enforcing linear system dynamics.[15]

2.1. The neural network is trained to minimize the reconstruction error between the original state and the decoded state after the linear evolution in the latent space, which is facilitated by the learned linear transformation $K(\Lambda)$ that acts on the encoded state y_t, resulting in the forward propogated state y_{t+1}.

The general form of the deep learning approach to learn the Koopman eigenfunctions can be described as follows. Given a dataset of state trajectories $x(t)$, the goal is to learn an encoding function $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ and a decoding function $\phi : \mathbb{R}^m \to \mathbb{R}^n$, such that the following holds:

$$\phi(\varphi(x_t)) \approx x_t. \tag{2.4}$$

$$\phi(K\varphi(x_t)) = \phi(\varphi(x_{t+1})). \tag{2.5}$$

Here, $\varphi(x_t)$ maps the state x_t to the latent space, y_t, and ϕ maps the latent representation back to the original state space. By training the neural network to minimize the reconstruction error, the encoder function φ effectively learns to approximate the Koopman eigenfunctions. Along with the eigenvalues, $K(\Lambda)$, produced from the auxiliary network Λ, to propogate forward in time the dynamics of the latent space, provide a powerful tool for the analysis and prediction of nonlinear dynamical systems.

2.3 Koopman Operator Meets Linear Control

The Koopman operator has been increasingly utilized for the analysis and control of nonlinear dynamical systems, as it allows for the construction of a linear representation of these systems. This linear representation not only enables the application of well-established linear control techniques, such as Linear Quadratic Regulator (LQR), to inherently nonlinear systems [13], but also greatly simplifies the implementation of Model Predictive Control (MPC). In the context of MPC, utilizing a linear model representation significantly reduces computational complexity, making the optimization problem easier and faster to solve.

One approach to control synthesis for nonlinear systems using the Koopman operator involves first finding a linear representation of the controlled system dynamics, and then applying MPC to this linear representation. Korda and Mezić [13] proposed a method using Extended Dynamic Mode Decomposition (eDMD) to approximate the Koopman operator for controlled systems, and subsequently performing MPC on the linearized system. The main challenge with this method lies in the selection of appropriate basis functions for eDMD, which requires prior knowledge of the system's dynamics.

Another approach was proposed by Proctor et al. [21], who used diffeomorphisms to identify the controlled model in the lifted space, and then performed MPC on the linearized model. This method is particularly tailored for systems governed by Lagrangian dynamics, and its application is limited for systems involving Eulerian dynamics, such as fluid flow.

Recently, deep learning has been employed to learn the eigenfunctions of the Koopman operator directly from data, offering a more flexible and data-driven approach [15]. This approach can be extended to account for control inputs in the neural network architecture, enabling the application of MPC on the linear representation obtained by the neural network. The resulting encoder function serves as an approximation of the Koopman eigenfunctions, and the linear dynamics in the latent space can be utilized for control synthesis, while ensuring convexity of the optimization problem associated with the control policy.

Formally, given a controlled dynamical system of the form $x_{t+1} = Ax_t + Bu_t$, where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^p$ is the control input, and $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$ represents the nonlinear system dynamics, the conventional MPC optimization problem is often non-convex due to the nonlinear nature of f, posing significant challenges for real-time control:

$$\begin{aligned}
&\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} l(x_k, u_k) + m(x_N) \\
&\text{subject to:} \\
&x_{k+1} = f(x_k, u_k), \quad k = 0, \dots, N-1 \\
&x_0 = x(t) \\
&u_k \in \mathcal{U}, \quad k = 0, \dots, N-1,
\end{aligned} \tag{2.6}$$

where N is the prediction horizon, $l(x_k, u_k)$ is the stage cost, $m(x_N)$ is the terminal cost, and $\mathcal{U}$ represents the control constraints.

However, by employing the Koopman operator framework, we seek to find a linear representation of the system dynamics in the lifted space, such that $y_{t+1} = Ay_t + Bu_t$, where $y \in \mathbb{R}^m$ is the transformed state, and $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{m\times p}$ are the system matrices in the lifted space. In this lifted space, the MPC optimization problem becomes:

$$\begin{aligned}
&\min_{u_0,\dots,u_{N-1}} \sum_{k=0}^{N-1} l(y_k,u_k)+m(y_N)\\
&\text{subject to:}\\
&y_{k+1}=Ay_k+Bu_k,\quad k=0,\dots,N-1\\
&y_0=\varphi(x(t))\\
&u_k\in\mathcal{U},\quad k=0,\dots,N-1,
\end{aligned} \tag{2.7}$$

where y_k is the state in the lifted space at step k, $\varphi:\mathbb{R}^n\to\mathbb{R}^m$ is the encoding mapping the original state space to the lifted space, $l(y_k,u_k)$ is the stage cost, and $m(y_N)$ is the terminal cost in the lifted space, with $\mathcal{U}$ remaining as the control constraints. This transformation results in a convex optimization problem, which simplifies the computational task and enhances the applicability of MPC for real-time control.

By using the Koopman operator-based linear representation of the controlled system, the nonlinear optimization problem associated with MPC can be transformed into a convex optimization problem, allowing for the efficient computation of globally optimal control policies. This approach has the advantage of overcoming the limitations of traditional Koopman operator approximations and offers improved prediction and control performance. Moreover, it eliminates the need for prior knowledge of the system's dynamics, as the Koopman eigenfunctions are learned directly from data.

However, it is important to note that the success of the Koopman-based MPC approach largely depends on the quality of the linear representation obtained for the controlled system. In cases where the Koopman operator approximation is poor, the resulting control policy may not perform well on the original nonlinear system. Therefore, further research is needed to improve the approximation techniques and develop more robust control synthesis methods that take into account the potential uncertainties in the linear representation.

Korda and Mezić's Methodology Overview

Korda and Mezić's approach in [13] describes some of the state of the art research in applying Koopman operator to linearly control non-linear dynamical systems, this requires extending the Koopman operator to controlled systems. The primary steps in their methodology are:

1. **Extended State Space**: Control inputs are incorporated into the state space, resulting in an extended state space that encompasses both states and control inputs.

   Given the original state space of the system as $x\in\mathbb{R}^n$ and the control inputs as $u\in\mathbb{R}^p$, the extended state space is defined as:

$$\tilde{x}=\begin{bmatrix}x\\u\end{bmatrix}\in\mathbb{R}^{n+p} \tag{2.8}$$

This extended state space $\tilde{x}$ is then used in the approximation of the Koopman operator, enabling the application of linear control techniques to the inherently nonlinear system.

2. **Koopman Operator Approximation**: Utilizing Extended Dynamic Mode Decomposition (eDMD), they approximate the Koopman operator for this extended state space. The Koopman operator here linearly evolves observables (functions of the state) even for nonlinear systems.

   Given the extended state space $\tilde{x}$ which includes both the state and control inputs, the Koopman operator $\mathcal{K}$ is approximated using eDMD. The operator acts on observables $g(\tilde{x})$, which are functions of the extended state space:

   $$\mathcal{K}g(\tilde{x}) = g(\mathcal{F}(\tilde{x})) \tag{2.9}$$

   Here, $\mathcal{F}$ represents the extended dynamical system, and $\mathcal{K}$ linearly evolves these observables even though the underlying system dynamics $\mathcal{F}$ are nonlinear.

3. **Model Predictive Control (MPC)**: The linearized dynamics obtained through the Koopman operator are then used in MPC. This linear representation allows the formulation of the MPC problem as a convex optimization, akin to linear systems.

This methodology offers a novel way to apply linear control techniques, like MPC, to nonlinear systems by leveraging the linearizing capabilities of the Koopman operator, but require the knowledge of lifting functions (or observables g) mentioned in [24], here we aim to learn this function, namely φ.

In summary, the application of the Koopman operator to controlled dynamical systems has shown promising results, enabling the use of linear control techniques for nonlinear systems. The main challenge lies in obtaining accurate linear representations of the controlled system dynamics and designing efficient control algorithms that exploit the benefits of the Koopman framework while addressing its limitations.

Chapter 3

System Identification & Control

3.1 4D Extended Dubins Car Model

4D extended Dubins car model as a controlled system was selected as a baseline to represent whether a simple system with fully known nonlinear dynamics could be captured and controlled using our methodology. The 4D extended Dubins car model is a widely used kinematic model for wheeled vehicles in control and robotics applications, especially pertinent to path planning, motion control, and obstacle avoidance tasks. This model simplifies the representation of a vehicle's motion by capturing its essential nonlinear dynamics.

The state of the vehicle is defined by the vector

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \end{bmatrix} \tag{3.1}$$

where x and y denote the coordinates of the vehicle's center of mass in a global reference frame, θ is the vehicle's heading angle, and v is its longitudinal velocity.

The evolution of the state is described by the following set of differential equations:

$$\dot{x} = v\cos(\theta), \tag{3.2}$$

$$\dot{y} = v\sin(\theta), \tag{3.3}$$

$$\dot{\theta} = \omega, \tag{3.4}$$

$$\dot{v} = a. \tag{3.5}$$

The control inputs for the system are the angular velocity ω and the longitudinal acceleration a, which govern the changes in orientation and speed of the vehicle, respectively. Here, ω represents the vehicle's turning rate, while a is the rate of change of the vehicle's velocity. Thus, the state transitions, defined as $\dot{x}, \dot{y}, \dot{\theta}, \dot{v}$, are influenced directly by the control inputs ω and a.

3.1.1 Data Generation

The generation of a high-quality dataset is crucial for the development and validation of control algorithms. In this study, we focus on the construction of a dataset that encapsulates feasible trajectories for the 4D extended Dubins car model. This process involves leveraging the concept of differential flatness, which allows for a systematic approach to trajectory planning by transforming the nonlinear dynamics into an equivalent linear representation.

Differential flatness is a property of some nonlinear control systems whereby all states and inputs of the system can be expressed as algebraic functions of a set of flat outputs and a finite number of their derivatives. A system is said to be differentially flat if there exists a set of outputs, known as flat outputs, which can be used to express the state and input variables of the system. Moreover, these flat outputs and a finite number of their derivatives are sufficient to capture the full dynamics of the system. Mathematically, for a general nonlinear system, if it is differentially flat, we can describe its states $\mathbf{x}$ and inputs $\mathbf{u}$ as:

$$\mathbf{x} = F_x(\eta, \dot{\eta}, \ddot{\eta}, \dots), \tag{3.6}$$
$$\mathbf{u} = F_u(\eta, \dot{\eta}, \ddot{\eta}, \dots), \tag{3.7}$$

where η denotes the flat outputs, and F_x and F_u are functions that map these flat outputs and their derivatives to the system's state and input vectors, respectively.

Applying this to the dynamics of the 4D extended Dubins car model, we consider the system's states $\mathbf{x}$ and inputs $\mathbf{u}$ as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ v \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \omega \\ a \end{bmatrix} \tag{3.8}$$

For this particular system, the flat outputs are the position coordinates x and y. The rest of the states and inputs can be derived from these flat outputs:

$$\theta = f_\theta(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dots) = \arctan 2(\dot{y}, \dot{x}), \tag{3.9}$$

$$v = f_v(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dots) = \begin{cases} \frac{\dot{x}}{\cos(\theta)} & \text{if } \cos(\theta) \neq 0 \\ \frac{\dot{y}}{\sin(\theta)} & \text{if } \cos(\theta) = 0 \end{cases}, \tag{3.10}$$

$$\omega = f_\omega(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dots) = \frac{\ddot{y} \cdot \dot{x} - \ddot{x} \cdot \dot{y}}{v^2}, \tag{3.11}$$

$$a = f_a(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, \dots) = \frac{\dot{x} \cdot \ddot{x} + \dot{y} \cdot \ddot{y}}{v}. \tag{3.12}$$

These functions, f_θ, f_v, f_ω, and f_a, map the flat outputs and their derivatives to the system's orientation θ, velocity v, angular velocity ω, and acceleration a. The explicit form of these functions is derived based on the dynamics of the 4D extended Dubins car and the constraints imposed by its physical capabilities.

If such functions exist, the 4D car model is differentially flat. By selecting suitable smooth functions for the flat outputs $x(t)$ and $y(t)$ that satisfy the initial and final positions of the car, we can generate feasible trajectories for the entire system. These trajectories respect the car's dynamics and constraints since they are derived from the flat outputs and their derivatives.

In this project, the dataset for the 4D extended Dubins car model is generated using Bezier curves, exploiting the differential flatness property. The Bezier curves are parameterized by a variable t within the range [0, 1], and for each trajectory, the control points are selected to ensure compliance with the car model's dynamics and constraints.

The dataset is generated by solving a linear system of equations using the constraints, x_c and y_c, defined in the flat output space as described in algorithm 1. The resulting trajectories are then used to compute the state and control trajectories for the car model.

In the context of Bezier curve trajectory generation, M is a matrix representing the basis functions of the Bezier curve, structured for the constraints at specific points along the trajectory. b_0 and b_1 are vectors of coefficients for the Bezier curves in the x and y dimensions, respectively. The system of equations to solve for these coefficients is given by:

$$M \cdot b_0 = x_c, \tag{3.13}$$

$$M \cdot b_1 = y_c. \tag{3.14}$$

These equations ensure that the resulting Bezier curves adhere to the specified constraints in the x and y dimensions.

Bezier Curve and Dataset Generation:

- **Trajectories**: We synthesized 70 trajectories with Bezier curves, parameterized by $t \in [0, 1]$. Each trajectory comprises 1000 data points, n, resulting in a total dataset of 70,000 points.

- **Constraints**: Each Bezier curve adheres to constraints within the Cartesian coordinates $x, y \in [-1.5, 1.5]$, ensuring the feasibility of the trajectories in accordance with the car model's dynamics.
- **Data Generation**: For each value of t, the positions $x(t)$ and $y(t)$ were computed, alongside other state and control variables, derived using the differential flatness property.

Algorithm 1 Generating Dataset using Differential Flatness

```
1: while $n > 1$ do
2:     Define $x$ and $y$ constraints along with matrix $M$
3:     Solve for coefficients of basis functions $b0$ and $b1$
4:     Compute $x$, $y$, $\dot{x}$, $\dot{y}$, $\ddot{x}$, and $\ddot{y}$ from $b_0$ and $b_1$
5:     Compute $\theta$ and $v$ trajectories
6:     Correct $v$ when $\cos(\theta) \approx 0$
7:     Compute control trajectories $\omega$ and $a$
8:     Update $n$
9: end while
```

3.2 Cart Pole

3.2.1 Dynamics

The Cart Pole problem is a classic control theory problem. The problem consists of a pole mounted on a cart that can move along a horizontal track. The pole is attached to the cart with a pivot point that allows the pole to rotate freely. The goal is to balance the pole vertically by applying forces to the cart. The system is inherently unstable, and the challenge is to design a controller or learn a policy that can keep the pole upright while possibly keeping the cart within certain bounds or moving it to a desired position.

In the Cart Pole problem, the system is typically described by a set of nonlinear differential equations that represent the dynamics of the cart and the pole. The state of the system consists of four variables: cart position, cart velocity, pole angle (measured from the vertical axis), and pole angular velocity. The control input is the horizontal force applied to the cart.

The states of the system are defined as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{3.15}$$

The given equations of motion are

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2 sin\theta = F \tag{3.16}$$

$$(I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta \tag{3.17}$$

The system dynamics for (v, ω) are implicitly written as:

$$\begin{bmatrix} M+m & ml\cos\theta \\ ml\cos\theta & I+ml^2 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -bv + ml\omega^2\sin\theta + F \\ -mgl\sin\theta \end{bmatrix} \tag{3.18}$$

Therefore the discretized dynamics are given as a system of equations, $a * \mathbf{x} = b$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ m+M & 0 & 0 & Mlcos(x_3) \\ 0 & 0 & 1 & 0 \\ 0 & mlcos(x_3) & 0 & I+Ml^2 \end{bmatrix} * \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_2 \\ -dx_2 + mlx_4^2 sin(x_3) + u \\ x_4 \\ -mglsin(x_3) \end{bmatrix} \tag{3.19}$$

3.2.2 Data Generation

Cart Pole data was generated using a Nonlinear programming (NLP) with the multiple shooting method. Using an NLP to solve for trajectories, given random initial states, allows us to create feasible control trajectories to reach the desired final state. The following Dynamics and values for variables were used.

$M = 0.5$
$m = 0.2$
$l = 0.3$
$I = 0.006$
$d = 0.1$
$g = 9.81$

The NLP control problem is defined as the following:

$$\begin{aligned} &\min \int_{t=0}^{T} F^2(t)\,dt \\ &\text{s.t} \\ &\dot{x} = f(x,u) \\ &|F(t)| \leq u_{bound} \\ &|X_1| \leq 1 \\ &x(0) = [x_{1(0)}, 0, x_{3(0)}, 0] \\ &x(T) = [x_{1(T)}, 0, \pi, 0] \end{aligned} \tag{3.20}$$

Where for this specific experiment the setting were T = 5, $u_{bound} = 2$, $x_1(0) \in [-1, 1]$, $x_1(T) \in [-1, 1]$, and $x_3(0) \in [-0.125\pi, 0.125\pi]$. Algorithm 2 defines the process for generating the dataset.

Algorithm 2 Cart Pole Data Generation

```
1: Define time horizon $t$, number of control intervals $N$
2: Requre dynamics from 3.19
3: Solve the NLP stated in 3.20
4: Extract optimal values for $x$ and $u$
```

3.3 Oil & Gas transport Pipeline

Pipelines play a vital role in the transportation of various resources such as clean water, natural gas, and liquid hydrocarbons across long distances, ensuring safe and efficient delivery [19]. In the context of oil and gas pipelines, pump stations equipped with variable frequency drives (VFDs) are crucial in maintaining the desired flow rate in accordance with the given schedule. Operators adjust the pumps to meet the demand, and machine learning approaches can be employed to analyze large datasets of pipeline historians, identifying trends and detecting anomalies. Consequently, automated real-time solutions can be developed to optimize overall costs and minimize carbon footprint [18].

We are working with an energy company's pipeline and pumping facilities in Northeastern Alberta, which is designed to support its Oil Sands Plant. The 16-inch Oil Sands Pipeline (OSPL) transports high-quality crude products and diesel fuel to markets in or near Edmonton, Alberta, connecting to other pipelines and markets across Canada and the U.S. The system's maximum operating capacity is estimated to be 1060 m3/hr when fully powered, sufficient to handle the additional product delivered from the Oil Sands Plant. The seven pump stations along the OSPL pipeline, named HS, WR, GL, NB, and SF, are capable of transporting natural gas liquids (NGL) extracted by the Olefins Project to the Fractionation Facility [20].

Our approach showcases the application of deep learning techniques for system identification and optimization of the pipeline operations. By learning the underlying dynamics of the system, we can provide optimal solutions for meeting the schedule while minimizing power consumption. This not only results in cost savings for the company but also contributes to reducing the environmental impact of the operation. The proposed method offers an innovative approach to managing complex pipeline systems, demonstrating the potential benefits of integrating machine learning and control theory in real-world applications [7].

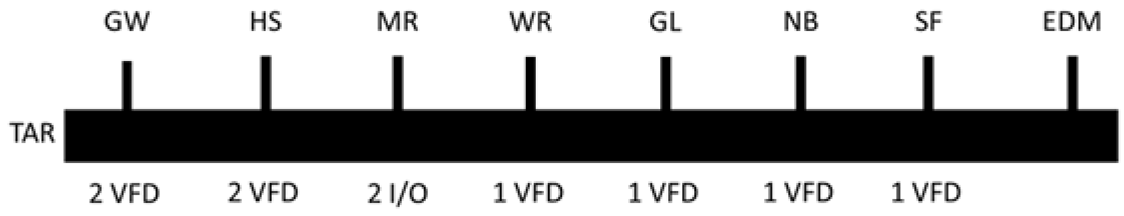

Figure 3.1: Structure of gas pipeline stations and corresponding VFD's

3.3.1 Problem Statement

The operation and management of pipeline systems, particularly those involving the transportation of valuable resources such as natural gas and liquid hydrocarbons, pose a unique set of challenges for operators. In the context of the generic energy company's pipeline and pumping facilities in Northeastern Alberta, operators must deal with several key aspects of the system to ensure efficient and cost-effective transportation.

The operators are responsible for managing the flow rates across the pipeline in accordance with the given schedule. This schedule consists of specific flow rate targets that must be met at various points along the pipeline. The operators receive the schedule and other relevant information, such as the pressure, temperature, and viscosity of the transported fluids, as well as the environmental conditions and external factors that may impact the pipeline operation. The operators must then adjust the pumps at each pump station, which are equipped with variable frequency drives (VFDs), to achieve the desired flow rates.

The challenges faced by the operators in managing the pipeline system stem from the complexity and nonlinearity of the underlying dynamics, as well as the uncertainty and variability of the external factors. Furthermore, the large-scale nature of the pipeline systems often makes it difficult for the operators to make real-time adjustments and respond quickly to the changing conditions. The lack of precise, real-time information and the need for manual intervention increase the risk of suboptimal decisions, resulting in increased costs and environmental impacts.

In light of these challenges, there is a clear need for innovative approaches that can facilitate the management of pipeline systems by providing the operators with accurate, real-time information and automated decision-making tools. Such approaches should be capable of handling the complex dynamics of the pipeline systems, accounting for the uncertainty and variability of the external factors, and optimizing the overall operation in terms of cost, efficiency, and environmental impact.

3.3.2 Data from Historian

Historian and Supervisory Control and Data Acquisition (SCADA) systems are essential tools for monitoring, controlling, and managing industrial processes and infrastructure.

SCADA systems are responsible for real-time data collection, process control, and communication between different devices and equipment, while historian systems store and manage the historical data generated by SCADA systems. The data collected by these systems is typically time-stamped and can be used for trend analysis, performance evaluation, and decision-making.

The data format in historian and SCADA systems often consists of a time series, with data recorded at regular intervals. In the case of the pipeline system, data is collected every 1 second. This high-resolution data allows for accurate monitoring and analysis of the system's behavior, enabling operators to make informed decisions and optimize the pipeline's operation.

The state and control variables for the pipeline system can be represented in the following table:

Table 3.1: State and control variables for the pipeline system

State Variables	**Control Variables**
Flowrate	HS Frequency
	WR Frequency
	GL Frequency
	NB Frequency
	SF Frequency

The state variable in this pipeline system is the flowrate, which represents the volume of fluid transported through the pipeline per unit of time. The control variables are the frequencies of the pumps at each pump station, namely HS, WR, GL, NB, and SF. By adjusting these frequencies, the operators can control the flowrate and meet the desired schedule while optimizing the system's performance in terms of energy consumption and environmental impact.

3.4 Network Architecture

The method for formulating a general representation of the Koopman Operator is presented in [15]. This formulation is of a general Neural Network architecture that enables learning of a nonlinear *uncontrolled* dynamical system. Here, we expand upon the ideas presented in [15] to propose a Neural Network architecture that is amenable to globally linearizing a controlled system.

3.4.1 Latent Space Linear System Parametrization

Jordan Canonical Form:

Instead of working with an arbitrary matrix A to describe the system dynamics, we employ the Jordan canonical form, which can be obtained from any matrix A through a similarity transform. The Jordan canonical form, denoted here as $K(\lambda)$, is the matrix A, formed as a block diagonal matrix composed of Jordan blocks J_i. The block diagonal matrix $K(\lambda)$ is defined as:

$$K(\lambda) = \begin{bmatrix} J_1 & 0 & \dots & 0 \\ 0 & J_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & J_k \end{bmatrix}, \tag{3.21}$$

where each Jordan block J_i either corresponds to a real eigenvalue or a pair of complex conjugate eigenvalues. For real eigenvalues, λ_i, Jordan blocks will be of the form:

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \dots & 0 \\ 0 & \lambda_i & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \lambda_i \end{bmatrix}. \tag{3.22}$$

while complex conjugate eigenvalues, $B(\mu, \omega)$, and I_2, the identity matrix of size 2x2, will be of the form:

$$\begin{bmatrix} B_i(\mu,\omega) & I_2 & \cdot & \dots & 0 \\ 0 & B_i(\mu,\omega) & I_2 & \dots & 0 \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & B_i(\mu,\omega) & I_2 \\ 0 & \cdot & \cdot & \dots & B_i(\mu,\omega) \end{bmatrix} \tag{3.23}$$

To account for this change, we adjust the Jordan block form [8] for complex eigenvalues as follows:

$$B\left(\mu,\ \omega\right) = \mu \begin{bmatrix} \cos(\omega) & \sin(\omega) \\ -\sin(\omega) & \cos(\omega) \end{bmatrix} \tag{3.24}$$

This representation is particularly powerful for systems analysis because it allows us to decouple the system into independent modes, each of which can be analyzed and controlled separately. In the context of our work, leveraging the Jordan canonical form simplifies the design of the Neural Network by providing a linear structure in the latent space that is reflective of the intrinsic dynamical properties of the system.

Full Encoder-Decoder Architecture

This architecture is a direct extension to the one presented in [15], and incorporates the control input of the dynamical system.

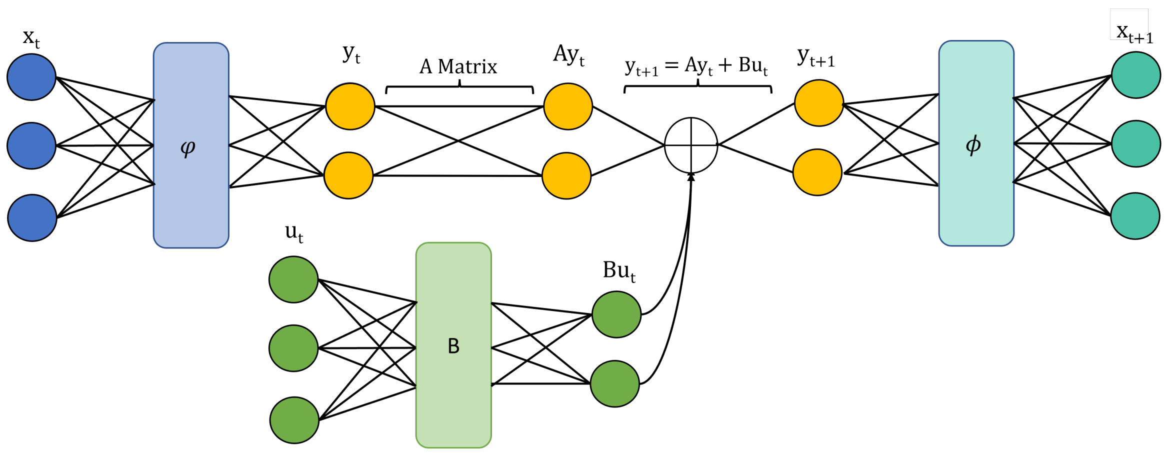

Figure 3.2: Proposed network architecture to identify controlled dynamical systems

Initially, the original state x_t is lifted to its extended state as $y_t = \varphi(x_t)$. y_t is then propagated through time by the $K(\lambda)$ matrix, which is an equivalent representation of A matrix in (3.25) giving us Ay_t. To achieve the true extended state evolved through time, y_{t+1}, we solve for it with the addition of the output from the B matrix: $y_{t+1} = K(\lambda)y_t + Bu_t$. The matrix represented by B is a single-layer network that has no activation function. The original state x_{t+1} is recovered through the decoder: $x_{t+1} = \phi(y_{t+1})$. This modified architecture allows us to have a full closed-loop solution for identifying linear models of controlled dynamical systems.
Training Procedure:
X_t, the original state is lifted to $y_t = \varphi(x_t)$, the extended state. y_t is fed through the learnt linear Dynamics A and B

$$y_{t+1} = Ay_t + Bu_t \tag{3.25}$$

To achieve the true extended state evolved through time, y_{t+1}. The original state x_{t+1} is recovered through the decoder: $x_{t+1} = \phi(y_{t+1})$.

Hyperparameters:

l : Layers in network $l \in \mathbb{R}$
w : Weight matrices
r : Number of real eigenvalues in lifted space $r \in \mathbb{R}$
im : Number of imaginary eigenvalues in lifted space $im \in \mathbb{R}$
S_p : Number of horizon shifts of state.
T : Number of horizon shifts middle of lifted state.
Tr : Trajectory length

Components:

Data (X): Training Data of total length n containing both state, S, and corresponding control sequences, U, for all trajectories. $S \in \mathbb{R}^{n \times s}$, $U \in \mathbb{R}^{n \times u}$.

Encoder (φ): Dimensions of Encoder weights in each of m layers, l_i, are defined by s, w, and k. $l_1 \in \mathbb{R}^{s \times w_1}$, $l_2 \in \mathbb{R}^{w_1 \times w_2}$, ..., $l_m \in \mathbb{R}^{w_{m-1} \times k}$

Decoder (ϕ): Dimensions of Decoder weights in each of m layers, l_i, are defined by s, w, and k. $l_1 \in \mathbb{R}^{k \times w_1}$, $l_2 \in \mathbb{R}^{w_1 \times w_2}$, ..., $l_m \in \mathbb{R}^{w_{m-1} \times s}$

Matrix A (A): A matrix of lifted linear space dynamics. $A \in \mathbb{R}^{k \times k}$. A is represented as a block diagonal matrix given $r, im.$ as defined in section blank.

Matrix B (B): B matrix of lifted linear space dynamics. $B \in \mathbb{R}^{k \times u}$

Following the extensions proposed to the framework by Lusch et al. [15], our loss functions integrate control actions into the learning process. The total loss function is a weighted sum of individual loss components, as shown in Equation 3.26.

- **Reconstruction Loss** ($\mathcal{L}_{recon}$): This loss term, defined in Equation 3.27, measures the accuracy of the Koopman autoencoder in reconstructing the state x. It ensures the mapping φ and its inverse ϕ are consistent, meaning the original state can be accurately reconstructed from its encoded representation.

- **Prediction Loss** ($\mathcal{L}_{pred}$): Described by Equation 3.28, this loss assesses the predictive capability of the Koopman operator. Over a prediction horizon of S_p, it compares the actual future states x_{m+1} with those predicted by applying the learned dynamics, which involves both the Koopman matrix K and control inputs u through the input matrix B.

- **Linearity Loss** ($\mathcal{L}_{lin}$): Presented in Equation 3.29, this term encourages the learned dynamics to be linear in the transformed space. It aims to minimize the difference between the Koopman operator applied iteratively to the initial encoded state and the encoded state at each subsequent time step.

The architecture facilitates closed-loop identification of controlled dynamical systems by learning to map the current state x_t to an extended state y_t using the lifting function φ, and then propagating this state through time using the learned dynamics matrix A and control input matrix B. The updated state y_{t+1} is a result of the Koopman operator applied to y_t and adjusted by the control inputs through B. The next original state x_{t+1} is then recovered by decoding y_{t+1} through ϕ.

The total loss is a composite measure that takes into account the reconstruction quality, the accuracy of predictions, the linearity of the learned dynamics, and the influence of control inputs. The hyperparameters α_1, α_2, and α_3 allow for balancing the importance of these different aspects during training. Moreover, the term $\|W\|_2^2$ in Equation 3.26 acts as a regularizer for the weights of the neural network to mitigate overfitting by penalizing large weights. The α setting for our testing were set at: $\alpha_1 = 1.0$, $\alpha_2 = 0$, and $\alpha_3 = 0.00001$

$$\mathcal{L}_{total} = \alpha_1(\mathcal{L}_{recon} + \mathcal{L}_{pred}) + \mathcal{L}_{lin} + \alpha_2\mathcal{L}_{inf} + \alpha_3\|W\|_2^2 \tag{3.26}$$

$$\mathcal{L}_{recon} = \|\,(x_1 - \phi(\varphi(x_1)))\,\|_2 \tag{3.27}$$

$$\mathcal{L}_{pred} = \frac{1}{S_p}\sum_{m=1}^{S_p} \|x_{m+1} - \phi(K^m\varphi(x_1) + \sum_{n=1}^{m} K^{n-1}Bu_{m-n+1})\|_2 \tag{3.28}$$

$$\mathcal{L}_{lin} = \frac{1}{T-1}\sum_{m=1}^{T-1} \|\varphi(x_{m+1}) - (K^m\varphi(x_1) + \sum_{n=1}^{m} K^{n-1}Bu_{m-n+1})\|_2 \tag{3.29}$$

Each term in the total loss function plays a crucial role in ensuring the model not only accurately represents the system dynamics but also responds appropriately to control inputs.

3.4.2 Training Procedure for System Identification Network

The algorithm for the system identification (System ID) network training is designed to learn a representation of the dynamical system which accounts for control inputs. The procedure iteratively updates the parameters of the network to minimize the reconstruction error and capture the dynamics accurately. The steps are as follows:

1. **Initialization**: The Encoder (φ), and the decoder (ϕ), and the matrices A and B are initialized with their respective weights. The dataset X is composed of state and control input pairs (x_t, u_t) and their subsequent states x_{t+1}.

2. **Data Preparation**: The training data x_{train} is formed from the trajectories in the dataset X and is batched according to a specified batch size.

3. **Training Loop**: The network enters a training loop which continues until the reconstruction error is below a threshold ϵ.

    - For the first 2 Epochs of training, the network parameters are updated with respect to only the reconstruction loss ($\mathcal{L}_{recon}$). This encourages the network to learn a faithful representation of the current state without considering future predictions or influence of control inputs.
    - After 2 Epochs have completed, the parameters are updated with respect to the total loss ($\mathcal{L}_{total}$) which incorporates the predictive capability with control input influence $\mathcal{L}_{pred}$, linearity enforcement $\mathcal{L}_{lin}$, along with the reconstruction error $\mathcal{L}_{recon}$.

- At each step, the total error, $\mathcal{L}_{total}$, is computed to monitor the training progress and serve as a stopping criterion if error is above a pre-set threshold.

Overall, the algorithm provides a structured method for the network to learn not just to reconstruct the current state, but also to predict the evolution of the system's state given control inputs, thus enabling it to model controlled dynamical systems effectively.

Algorithm 3 System ID network Training

```
Require: $l, w, r, im, S_p, T, Traj$
 1: $\varphi \leftarrow w_\varphi$
 2: $\phi \leftarrow w_\phi$
 3: $A \leftarrow w_A$
 4: $B \leftarrow w_B$
 5: $X \leftarrow \text{Data} = \{(x_t, u_t), x_{t+1}\}_{t=1}^{n}$
 6: $x_{\text{train}} \leftarrow$ X formed by $Tr$, $S_p$ and $T$
 7: while Error $\leq \epsilon$ do
 8:     if $epoch \leq 2$ then
 9:         $w_\varphi, w_\phi, A, B \leftarrow$ Updated $w_\varphi, w_\phi, A, B$ w.r.t $\mathcal{L}_{\text{recon}}$
10:     else if $epoch \geq 2$ then
11:         $\varphi, \phi, A, B \leftarrow$ Updated $w_\varphi, w_\phi, A, B$ w.r.t $\mathcal{L}_{\text{total}}$
12:         Error $\leftarrow \mathcal{L}_{\text{total}}$
13:     end if
14:     if Error $\geq \epsilon$ then
15:         Exit
16:     end if
17: end while
```

3.5 Control

Following the detailed exposition of the training process for the Koopman operator neural network, we transition to the practical application of the trained network in control tasks. The network, now equipped to approximate the complex dynamics of the system within a linear framework, forms the backbone of the control algorithm we are about to discuss. This section elucidates the steps involved in deploying the trained neural network to achieve effective control over the system's trajectory. The algorithm leverages the Koopman operator's linear representation of the nonlinear dynamics, enabling us to apply linear control techniques to inherently nonlinear systems. We provide a comprehensive walkthrough of the algorithm, ensuring a clear understanding of its execution and application in real-world control scenarios. Both 4D extended Dubins car and Cart Pole are controlled using this control methodology.

In algorithm 4, the procedure for controlling dynamical systems through a learned linear approximation of their behavior. The approach leverages the pre-trained neural network that models the system's dynamics via the Koopman operator, enabling the use of linear control techniques.

- **Initialization:** We begin by loading a trained neural network denoted as $\mathcal{N}$. This network has encoded within it the Koopman operator representation of the system's dynamics, allowing us to abstract the nonlinear system as a linear one in a higher-dimensional space.
- The mappings φ, A, and B are then extracted from $\mathcal{N}$. Here, φ represents the observable function, which maps the state x into the Koopman space. Matrices A and B define the linear dynamics in the Koopman space, corresponding to the state transition and control matrices, respectively.
- The **Trajectory** to be tracked is defined as $Track$, with the initial state x_0 set to the first element of this trajectory.
- The algorithm enters a loop that iterates over the length of the trajectory. For each iteration:
  - The reference output y_{ref} is determined for the current time step, extending over the control horizon.
  - The initial output y_0 in the Koopman space is obtained by applying the observable function φ to the current state x_0.
  - A Quadratic Program (QP) is solved to determine the control input u that minimizes the cost function, defined in terms of the reference trajectory, the current state, and the linear dynamics.
  - The true dynamics function f is then used to update the state x_i based on the current state x_0 and the control input u. This step accounts for the real-world application where the model's predictions are used to inform control actions.
  - The state variable x_0 is updated to the new state x_i, and the process repeats.
- The loop continues until the end of the trajectory is reached, thereby ensuring the system tracks the desired path as closely as possible.

This algorithm effectively combines the predictive power of neural networks with the efficiency of linear control methods. By doing so, it provides a robust approach to control complex, nonlinear systems in a computationally tractable manner.

NMPC Control For 4D Extended Dubins Car Model

Having delineated the linear control approach using the Koopman operator neural network, we will compare the performance with a Nonlinear Model Predictive Control (NMPC) of

Algorithm 4 Linear controller using Koopman Neural Network

```
1: Initialize: Load Trained Network: $\mathcal{N}$
2: $\varphi \leftarrow \varphi$ from $\mathcal{N}$
3: $A \leftarrow A$ from $\mathcal{N}$
4: $B \leftarrow B$ from $\mathcal{N}$
5: $Track \leftarrow$ Trajectory to be tracked
6: $x_0 \leftarrow Track(0)$
7: for $i \leftarrow 1$ to Trajectory length do
8:     $y_{ref} \leftarrow Track[i : i + \text{horizon}]$
9:     $y_0 \leftarrow \varphi(x_0)$
10:    $u \leftarrow$Solve QP$(y_0, y_{ref}, A, B)$
11:    $x_i \leftarrow f(x_0, u)$                    ▷ f(x, u) is the true dynamics
12:    $x_0 \leftarrow x_i$
13: end for
```

4D extended Dubins car. The objective is to highlight the differences in performance and efficiency between our proposed linear MPC and a more traditional NMPC approach, which utilizes the known system dynamics.

The Nonlinear Model Predictive Control (NMPC), described in algorithm 5 for the 4D extended Dubins car is designed to compute control inputs that minimize the deviation from a desired trajectory over a given prediction horizon.

The algorithm begins with the definition of the target trajectory, which the 4D extended Dubins car aims to follow, and a prediction horizon, which sets the timeframe for the trajectory optimization. The state x and control u sequences are initialized to prepare for the optimization process, which seeks to minimize a predefined cost function C that quantifies the difference between the predicted states and the target trajectory points.

An optimization problem is then formulated, wherein the cost function is subjected to the dynamics of the 4D extended Dubins car, represented by the function f. This function describes how the state of the car evolves over time in response to applied control inputs. The integral within the dynamic constraints captures the continuous nature of the system's evolution between discrete time points.

A numerical optimization solver, such as the Interior Point OPTimizer (IPOPT), is utilized to solve this problem. The solver iteratively adjusts the control sequence to find the minimum cost, respecting the dynamics and any constraints imposed on the states and controls.

Once the optimal control sequence is computed, it is applied to the 4D extended Dubins car model incrementally. At each step, the state of the car is updated using the dynamic function f, which ensures that the car's motion adheres to its physical capabilities and the applied controls. This process repeats for the length of the prediction horizon, continually adjusting the car's trajectory towards the target.

By iteratively solving this optimization problem and applying the resulting controls, the NMPC algorithm steers the 4D extended Dubins car along a path that closely follows the desired trajectory, effectively handling the system's nonlinear characteristics.

Algorithm 5 Nonlinear Model Predictive Control for 4D extended Dubins car

```
1: **Given:** A 4D extended Dubins car model, target trajectory $Tr$, and prediction horizon $N$.
2: **Objective:** Minimize the cost function $C$ tracking the trajectory.
3: Initialize state $X$ and control $u$ trajectories with dimensionality according to $N$.
4: Initialize the dynamics function $f$ representing the 4D extended Dubins car.
5: **Optimization Problem:** Formulate the NMPC as
6: $$\min_u C(x, u) = \sum_{k=1}^{N} \|X_k - Traj_k\|^2$$
   subject to
7: $$x_{k+1} = x_k + \int_{t_k}^{t_{k+1}} f(x_k, u_k)\, dt, \quad k = 1, \ldots, N-1$$
8: with appropriate initial conditions and state/control constraints.
9: Solve the optimization problem using a solver (e.g., IPOPT).
10: for $i \leftarrow 1$ to $N$ do
11:     Apply the optimal control $U_i$ to the 4D extended Dubins car car model.
12:     Update the state $X_i$ using the dynamics function $f$.
13: end for
```

3.6 Limitations and advantages of network design

3.6.1 Limitations

The method discussed presents a viable methodology for system identification of controlled dynamical systems. However, it is crucial to acknowledge its inherent limitations to fully understand its applicability and scope.

One of the primary limitations of this approach arises from the fact that state constraints are not directly accommodated by the lifted Koopman representation[5, 12]. The lifting process translates the nonlinear dynamics into a higher-dimensional linear space, which simplifies the computation but does not inherently account for constraints in the original state space[16]. For example, if we consider state constraints as they may be placed in equation 2.6 as $x_k \in \mathcal{X}, \quad k = 0, \ldots, N-1$, One state constraint can be given as $\mathcal{X}_1 \leq 0$, it is not straightforward to translate these into the lifted space as $\varphi(\mathcal{X}_1)) \leq 0$, since $\varphi(\mathcal{X}_1)$ does not necessarily preserve the structure of $\mathcal{X}_1$. This can be expressed as:

$$\phi(\mathcal{X}_1) \nleq 0 \Rightarrow \mathcal{X}_1 \nleq 0 \quad \forall x \in \varphi(x), \tag{3.30}$$

where (x) is in the original state space, and is insinuated by equation 2.4.

3.6.2 Benefits

Despite the aforementioned limitation, the Koopman-based system identification approach yields several benefits that make it a robust model, especially when integrated with reinforcement learning (RL) frameworks.

- **RL Model Training**: The system identification portion of this method can serve as an effective model for RL to train on[3], allowing RL algorithms to learn optimal control policies by leveraging the predictive accuracy of the linear Koopman model.
- **No Need for Exploration**: Unlike traditional RL which requires extensive exploration of the state space, the Koopman method[21] can identify system dynamics without the need for such exploration. This is due to the method's ability to lift the system into a space where linear dynamics suffice to describe the evolution of states over time.
- **Convexity and Efficiency**: The linearity of the Koopman model renders the problem convex[14], making it computationally efficient as convex problems have globally optimal solutions that can be found efficiently[2]. This efficiency is a stark contrast to solving non-linear models, which can be computationally intensive and may only guarantee local optima.
- **Analytical Clarity**: Linear models facilitate easier analysis and interpretation of the system's characteristics[1]. For instance, the eigenvalues of the learned dynamics matrix A provide insight into the system's stability, a property that is much more challenging to assess in non-linear systems[9].

$$\lambda(A) \Rightarrow \begin{cases} \text{stable} & \text{if } |\lambda_i| < 1\,\forall i \\ \text{unstable} & \text{otherwise} \end{cases} \tag{3.31}$$

In conclusion, while the Koopman-based approach to system identification has its limitations, particularly with state constraints, its benefits in terms of computational efficiency, ease of analysis make it a powerful tool for modeling and controlling dynamical systems.

Chapter 4

Results

Research Question and Objectives

This research seeks to answer a crucial question in the realm of control systems: Can non-linear systems be accurately modeled as linear systems, such that they can be effectively controlled using a linear Model Predictive Control (MPC) framework? The primary objectives of this study are twofold. First, to train neural network models on various systems, notably the 4D extended Dubins car, cart pole, and a gas pipeline, ensuring that these models can replicate the dynamics of the actual systems. Second, to validate the performance of these trained models under a linear MPC loop formulated as a Quadratic Programming (QP) problem.

Methodology Overview

To address the research question and achieve our objectives, the following methodology was adopted:

For both 4D extended Dubins car & Cart Pole Systems, true dynamics are available. We began by training multiple neural network models on these systems. Post training, control was performed via a linear MPC loop, formulated as a QP defined algorithm 4, and were tested for their capability to track a predetermined trajectory or target respectively. This served not only as a validation of the neural network model's accuracy but also showcased the viability of our overarching approach.

Gas Pipeline System

Unlike the 4D extended Dubins car and Cart Pole systems, the true dynamics for the gas pipeline are not available. However, given its real-world significance and complexity, it served as an essential testbed for our methodology. Neural network models were trained on this system, and their performance was evaluated based on the training and validation errors. Drawing parallels from the success observed in the 4D extended Dubins car and Cart Pole

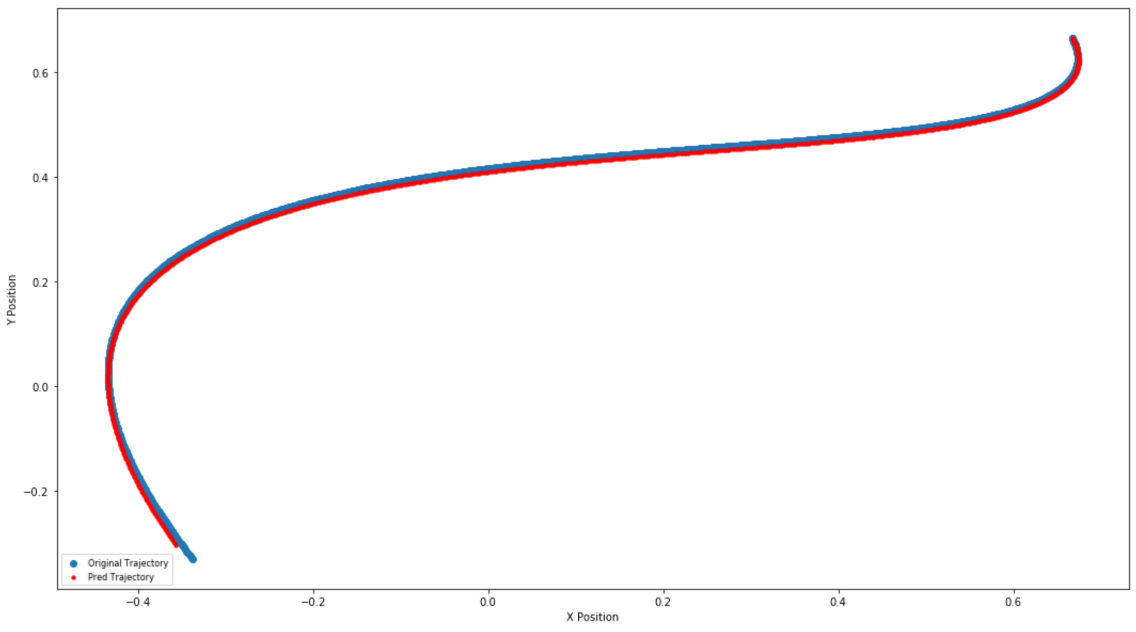

Figure 4.1: Trajectory tracked by controller using Koopman representation of 4D extended Dubins car

systems, this segment provides a projection of the real-world applicability and potential benefits of the proposed methodology.

With the methodologies set and systems analyzed, the subsequent sections will delve into the detailed results obtained from each system, providing insights, comparisons, and implications of the findings.

4.1 4D Extended Dubins Car

4.1.1 Network settings and MPC control results

The comparison between the desired trajectory and the trajectory resulted from the application of our proposed control method is illustrated in 4.1 and 4.2. Both figures show trajectories that are tracked by our proposed methodologies, however 4.1 was enlarged to showcase the precision of the tracking. The 'Original Trajectory' represents the target trajectory, while the "Predicted Trajectory" delineates the path computed by the linear MPC using the trained neural network model. The x and y axis describes the x and y position of 4D extended Dubins car respectively. The proposed control methodology exhibits promising performance by tracking the target trajectory with high fidelity. The plots visually demonstrate the efficacy of the control strategy, with the controlled tracking (in blue) closely following the target trajectory (in red) across different state variables like position (x, y), steering angle (theta), and velocity. The close adherence of the two paths underscores the model's capability in capturing the nonlinear dynamics of the 4D extended Dubins car.

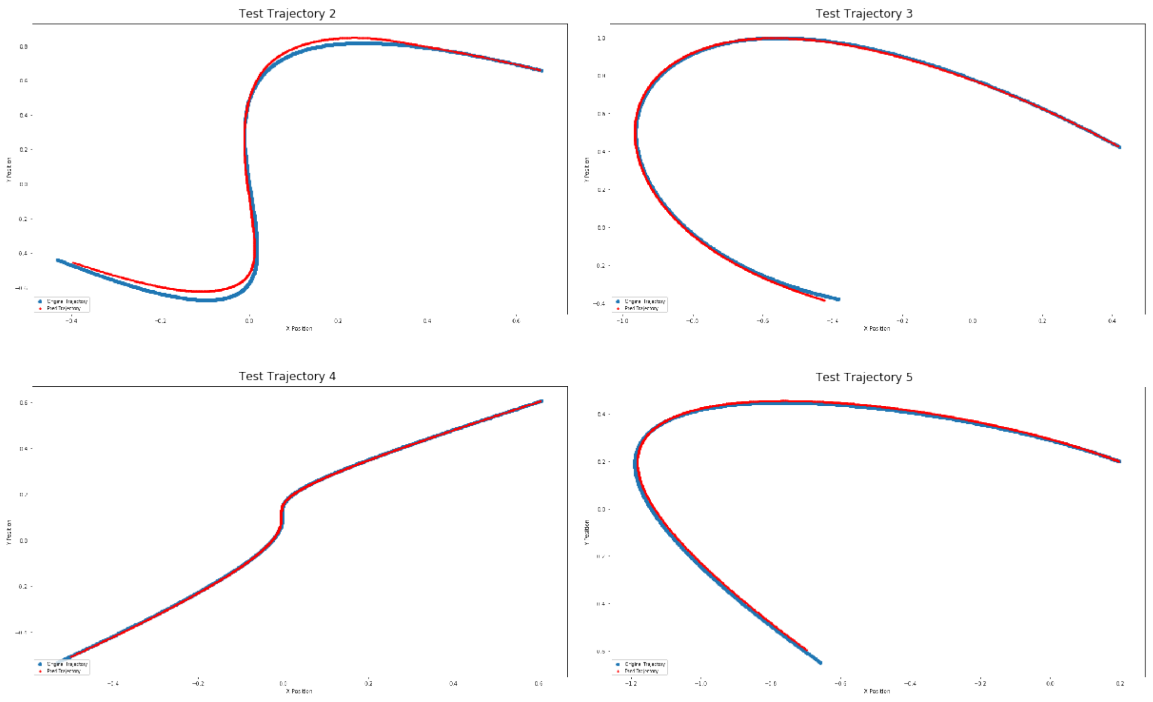

Figure 4.2: Other test Trajectories tracked by controller using Koopman representation of 4D extended Dubins car

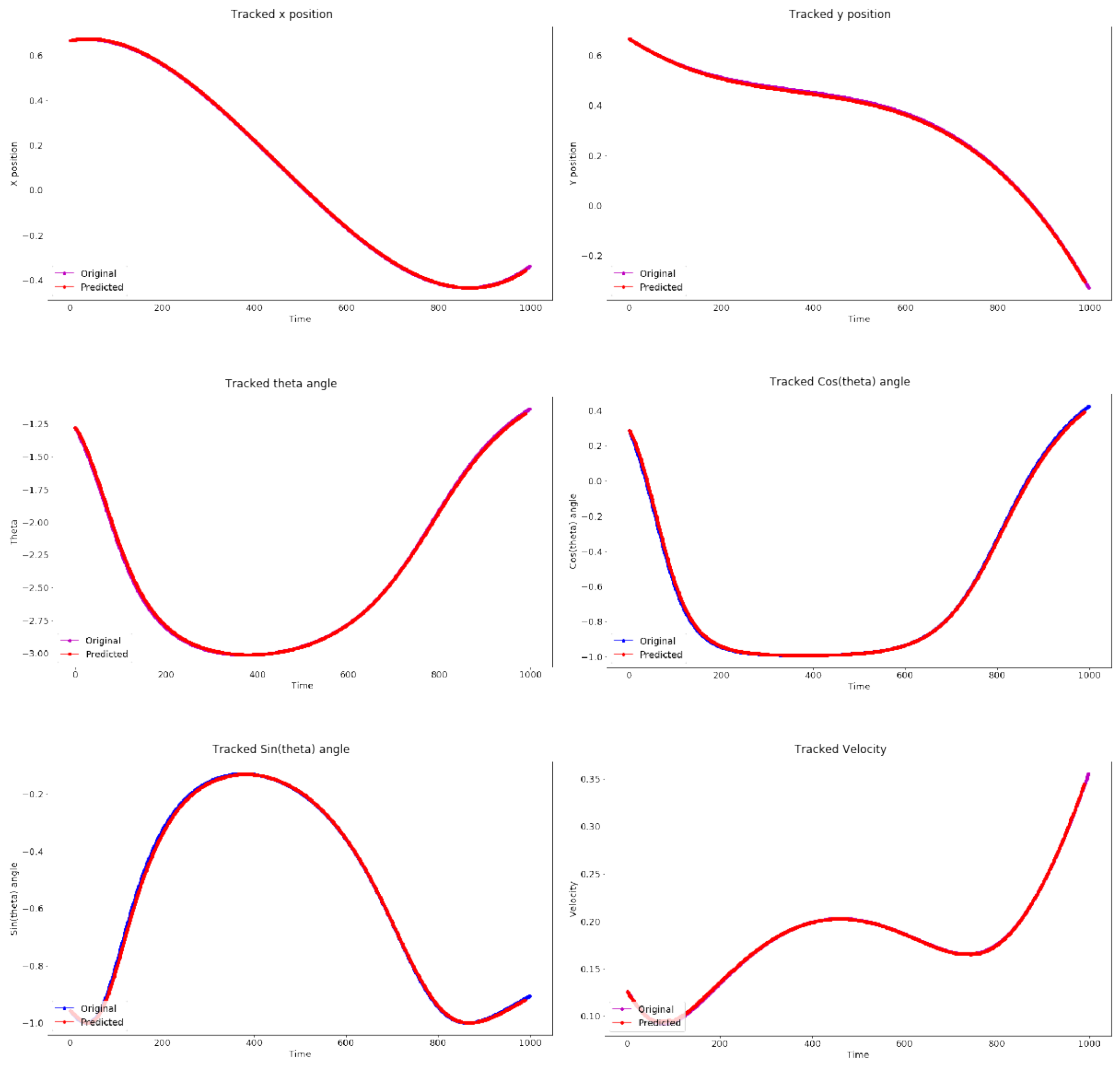

Figure 4.3: State variables tracked for trajectory shown in Figure 3.2

Further examination of individual state tracked for the control performed in Figure 4.1 presents a granular view of the controller's performance. The specific states for the tracked trajectory are shown in Figure 4.3. The graphs showcase the different states being tracked over time in the x-axis, and their respective value in the y-axis, revealing the nuanced behavior of each state and offering insights into the controller's precision and responsiveness to track each of the specific states.

The neural network was trained with the following set of parameters, listed in Table 4.1. The architecture was defined to accommodate the complexity of the system dynamics.

The error as described in equation 3.26 achieved during training was 8.70×10^{-5}, indicating the model's ability to learn the system dynamics accurately. The Mean Absolute Error (MAE) test error, computed as the average discrepancy across the tracked states and

Table 4.1: Neural network training parameters for 4D extended Dubins car.

Parameter	**Value**
S_p: Number of shifts	7
T: Number of shifts middle	7
r: Number of real	10
im: Number of complex pairs	10

the entire trajectory length, was 0.001323. This demonstrates a satisfactory generalization of the network to unseen data.

The results obtained suggest that the linearization approach for non-linear systems, when combined with neural network models, can be a robust method for controlling complex dynamics. The low error values, both in training and testing, coupled with the precise trajectory tracking exhibited by the 4D extended Dubins car, bolster the hypothesis that linear MPC frameworks can indeed manage non-linear systems effectively.

4.1.2 Linear controller with trained koopman network vs. NMPC

In the results presented, we observe a close parallel in the trajectory tracking performance between the Nonlinear Model Predictive Control (NMPC), shown in Figure 4.4 and our proposed linear control approach using the Koopman operator neural network shown in Figure 4.1, for the same target trajectory. Despite the NMPC's advantage of being furnished with the system's true dynamics, our linear controller demonstrates a remarkable ability to track the desired trajectory with comparable precision. The trajectories depicted in the attached figures reveal that the linear controller not only successfully approximates the nonlinear behavior of the system but also maintains a trajectory that is tightly aligned with that of the NMPC. This congruence is especially noteworthy considering the linear controller operates on a learned representation of the system, essentially a high-dimensional linear abstraction, rather than direct nonlinear modeling. The outcome underlines the efficacy of the Koopman operator framework in synthesizing complex dynamical behaviors and poses a compelling argument for its use as a viable alternative to traditional nonlinear control strategies in certain applications.

4.1.3 Linear controller with linear regression and polynomial basis functions

A baseline approach to modeling the dynamics of a 4D extended Dubins car is via polynomial basis functions is proposed. The methodology entails generating an enhanced dataset that encapsulates the nonlinear dynamics of the system through a polynomial transformations of the state variables.

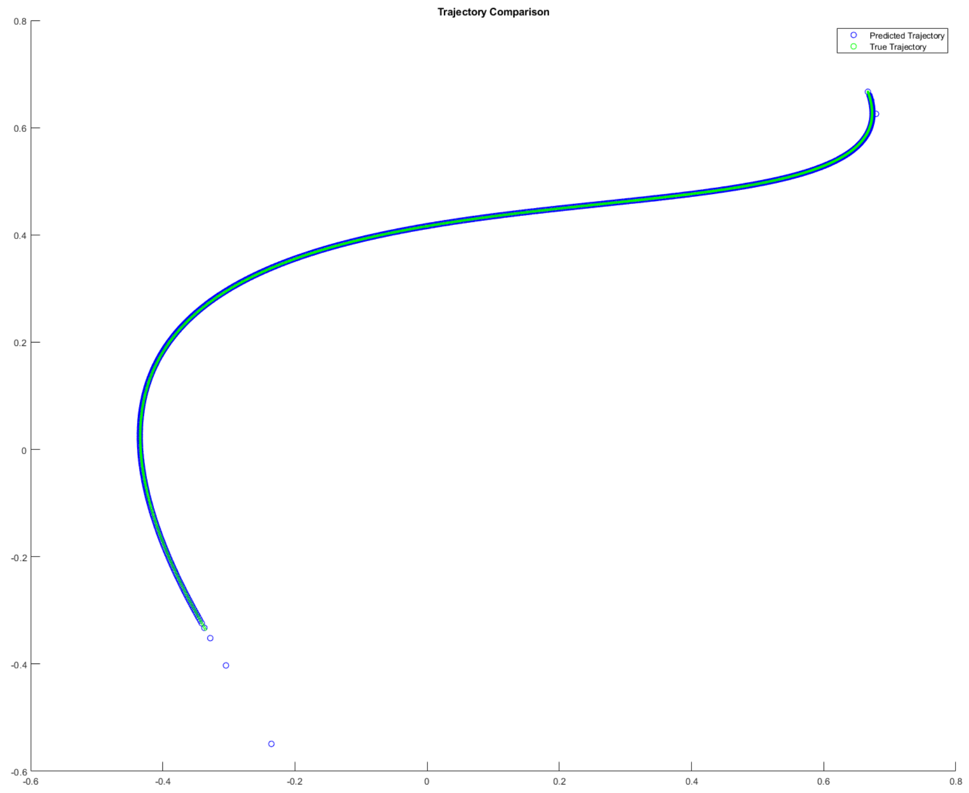

Figure 4.4: NMPC control of 4D extended Dubins car model with true dynamics given

Dataset Generation

Given the state, variables of the Dubins car: x, y, θ, and v, polynomial basis functions up to the third degree are constructed. Analogously to our methodology, these polynomial basis functions effectively act as our lifting function φ. This results in a set of features, each a polynomial combination of these state variables:

$$\begin{aligned}
\text{Original State Variables:}\quad & x, y, \theta, v \\
\text{0th Degree:}\quad & 1 \\
\text{1st Degree:}\quad & x, y, \theta, v \\
\text{2nd Degree:}\quad & x^2, y^2, \theta^2, v^2, xy, x\theta, xv, y\theta, yv, \theta v \\
\text{3rd Degree:}\quad & x^3, y^3, \theta^3, v^3, x^2y, x^2\theta, x^2v, xy^2, \dots \\
& \text{(and other combinations up to the third degree)} \in \mathbb{R}^{20}
\end{aligned}$$

These features form the new state, $z \in \mathbb{R}^{35}$, and are then used to train a linear regression model to predict the subsequent state variables one time step, $t+1$, ahead. Relative to our most successful implementation using our methodology, the lifted space $y \in \mathbb{R}^{20}$ is much smaller relative to $z \in \mathbb{R}^{35}$.

Model Training

The model is trained using a linear regression algorithm without an intercept to ensure that the zero state corresponds to zero control input. Given the enhanced state, z, and control data, u, the model is trained to predict the subsequent state variables.

$$\begin{aligned}
z &= \text{state data}, \\
u &= \text{control data}, \\
X &= \begin{bmatrix} z \\ u \end{bmatrix}
\end{aligned} \tag{4.1}$$

The linear regression model predicts the next state variables z_{t+1} based on the current state and control data, X. The model is defined as:

$$z_{t+1} = W \cdot X_t \tag{4.2}$$

The Mean Squared Error (MSE) loss function for training is defined as:

$$\text{MSE} = \frac{1}{N} \sum_{t=1}^{N} \|z_{t+1} - W \cdot X_t\|^2 \tag{4.3}$$

where:

- N is the number of data points.
- z_{t+1} is the actual next state.
- W is the weight matrix combining A and B.
- X_t is the combined state and control data at time t.

The objective of the training process is to minimize this MSE loss function, leading to the determination of the optimal matrices A and B.

The error as described in equation 4.3 achieved during training was 0.101, which is very high compared to the error calculated using our methodology: 8.70×10^{-5}. Validation error was 0.124.

Given the state data $z \in \mathbb{R}^n$, where n is 35 and the control data $u \in \mathbb{R}^p$, where p is 2, the combined data matrix X and the learned weight matrix W from the regression are represented as:

$$X = \begin{bmatrix} z \\ u \end{bmatrix}, \quad W = \begin{bmatrix} A \\ B \end{bmatrix} \tag{4.4}$$

Here, after training, the matrix W is partitioned into two matrices:

$$\begin{aligned} A &\in \mathbb{R}^{n\times n} \quad \text{(corresponding to state data } z\text{)}, \\ B &\in \mathbb{R}^{n\times p} \quad \text{(corresponding to control data } u\text{)}. \end{aligned}$$

The matrices A and B are used for the optimal control of the system.

Control Loop and Results

Upon deploying the trained model within the control loop described in equation 2.7, unexpected behavior was observed. The controller's outputs and the system's states exhibited rapid escalation. This outcome suggests potential issues in the model or control loop design, which can be attributed to several factors such as model overfitting, numerical instability, or discrepancies between the trained model's assumptions and the actual system dynamics.

The red dots in Figure 4.5 represents the resulting states generated by the controller. The system quicky becomes unstable and is not able to adhere to the trajectory to be tracked. The learned model fails to capture the dynamics of the system such that it may be used in a stable and robust manner under control. The divergence between the controller's outputs and the ideal trajectory underscores the shortcomings of a polynomial basis function based approach to modelling a system to be used under control. This shows a need for learning lifting functions for the Koopman operator using more robust methods.

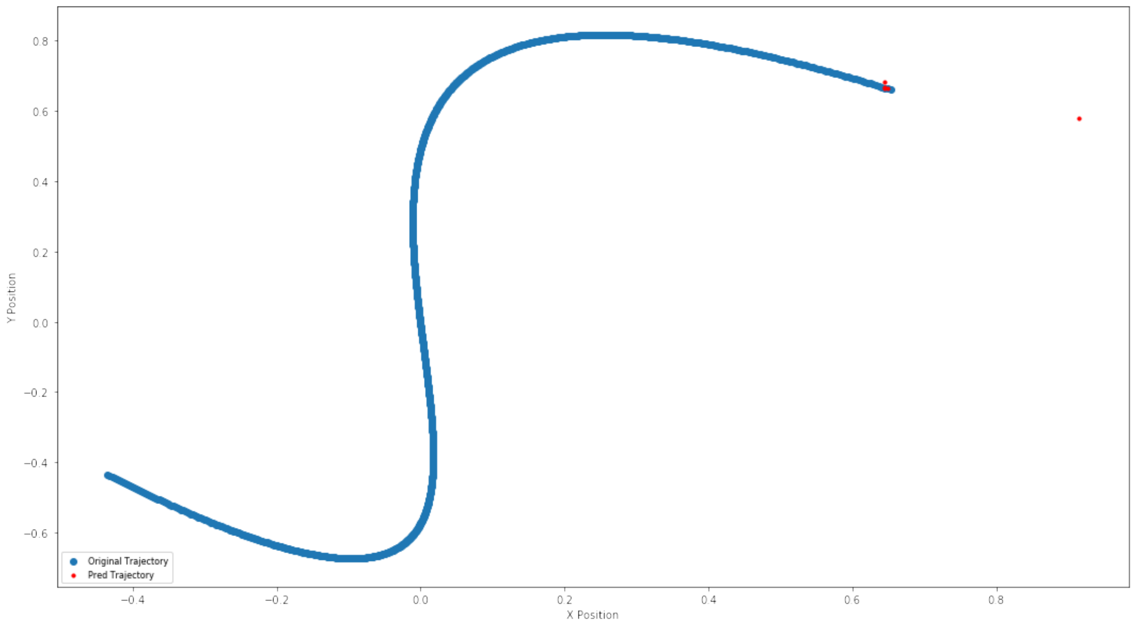

Figure 4.5: Comparison of Polynomial basis model controller outputs against the ideal trajectory.

4.1.4 Finding the correct network parameters for MPC

During the course of evaluating the performance of various neural networks within a Model Predictive Control (MPC) loop, several interesting insights emerged. The networks' capabilities were gauged based on their hyperparameter settings namely number of shifts, number of shifts middle, and latent space size which define the prediction horizon in real and lifted spaces, and the size of the lifted space, respectively.

The training phase results, depicted in Figure 4.6, illustrate a spectrum of training errors across various hyperparameter configurations. A key observation here is that a lower training error does not necessarily equate to superior performance during MPC testing. This is underscored by two particular networks (highlighted in green in 4.7), which, despite their differing training errors, both succeeded in satisfactorily tracking the target trajectory under MPC. Of these, one network emerged as superior; however, it exhibited a higher testing error in the MPC loop.

Furthermore, the hypothesis that networks trained with a longer prediction horizon (number of shifts and number of shifts middle) would inherently yield better MPC performance was not substantiated by the empirical evidence. This indicates that the ability to predict further into the future does not automatically translate to more effective control actions when integrated into the MPC framework.

A pivotal finding is that there seems to be an optimal region of hyperparameter configurations where the network not only learns effectively but also translates that learning into

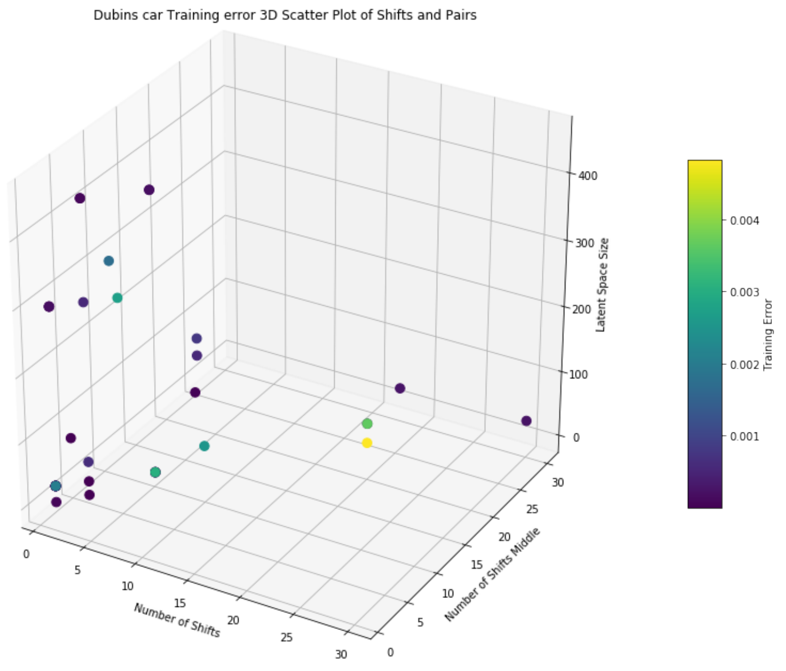

Figure 4.6: 3D plot of 4D extended Dubins car training runs showcasing Error vs. Hyperparameters.

successful trajectory tracking within the MPC. This sweet spot must be identified through empirical testing, as it is not readily apparent through training error metrics alone.

These insights emphasize the complexity of designing neural networks for control tasks. It is not sufficient to minimize training error; one must also consider the network's capacity to generalize to new situations, as encountered in the testing phase under MPC. This may involve a trade-off between the accuracy of predictions and the robustness of control performance.

4.2 Cart Pole

Two samples of the control tasks are shown below to validate the control methodology:

1. Bringing the pole to a vertical position ($\theta = 0$) while keeping the cart at the origin ($x = 0$), shown in Figure 4.9.

2. Bringing the pole to a vertical position ($\theta = 0$) at a specified cart position ($x = 1$), shown in Figure 4.8.

Both tasks begin with the system at rest, with initial conditions set to zero for all state variables. The error as described in equation 3.26 achieved during training for the model that was successful in performing control was 7.35×10^{-3}.

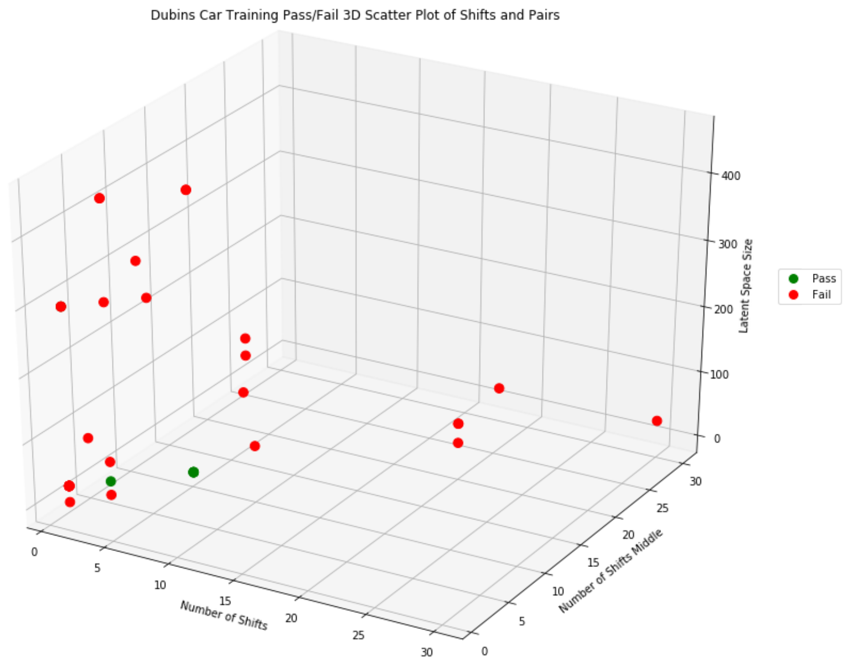

Figure 4.7: 3D plot of 4D extended Dubins car Testing runs based of pass/fail criteria under control

Figure 4.8 and Figure 4.9 illustrate the trajectories of the state variables during the control tasks. The results demonstrate the system's capability to reach and maintain the specified target states under the guidance of the MPC with the trained models.

The trajectory plots show that for both tasks, the control objectives are met successfully. In the first scenario, the cart remains stationary at the origin, while in the second, it moves to and stabilizes at the desired position of $x = 1$. These outcomes versatility and reliability of our control methodology in handling the nonlinear dynamics of the Cart Pole system.

The relavant parameters set are represented in Table 4.2.

Table 4.2: Neural network training parameters for 4D extended Dubins car.

Parameter	**Value**
S_p: Num shifts	10
T: Num shifts middle	10
r: Num real	30
im: Num complex pairs	30

4.2.1 Finding the correct network parameters for MPC

In the preceding analysis of the 4D extended Dubins car, we observed the nuanced relationship between training errors of neural networks and their subsequent performance under

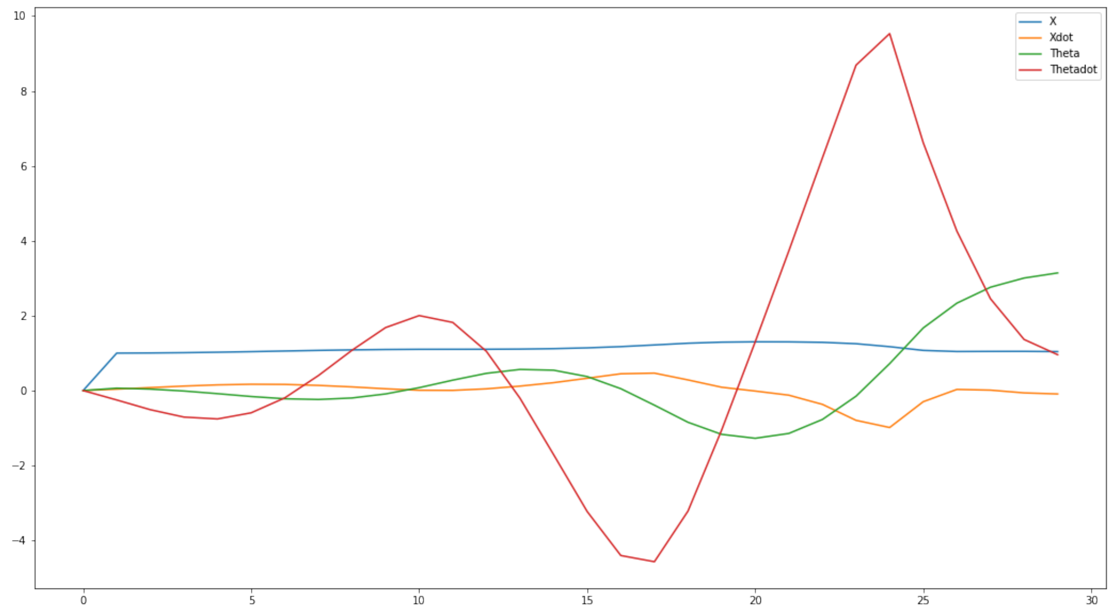

Figure 4.8: Trajectory tracked by controller using Koopman representation of Cart Pole to $x = 1$

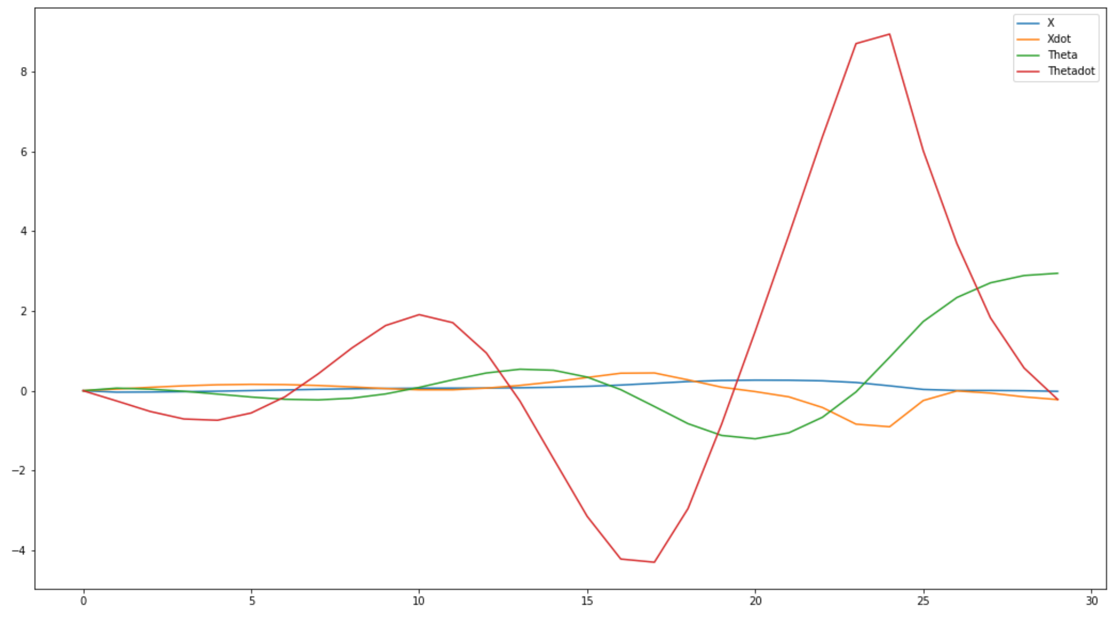

Figure 4.9: Trajectory tracked by controller using Koopman representation of Cart Pole to $x = 0$

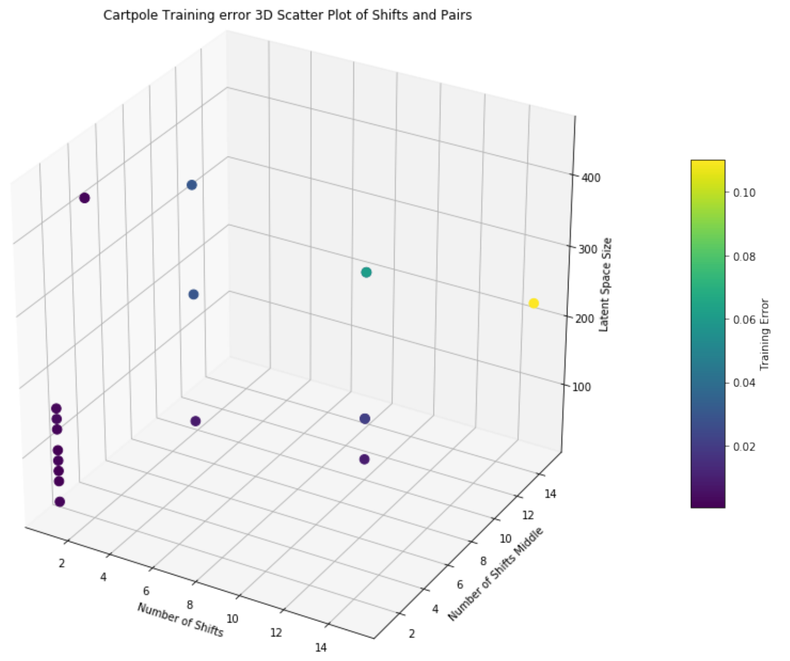

Figure 4.10: 3D plot of Cart Pole training runs showcasing Error vs. Hyperparameters

Model Predictive Control (MPC). This section extends the investigation to the Cart Pole system, to discern whether similar patterns emerge.

The Cart Pole results, consistent with our findings from the 4D extended Dubins car analysis, reinforce the assertion that a low training error is not a definitive predictor of efficacy in control tasks. And very large values for the hyperparameter also does not ensure effective control. The singular network that succesfully controlled the system did so possibly due to an alignment between its predictive capabilities and the dynamical intricacies of the Cart Pole system under the MPC regime.

These experiments, along with the 4D extended Dubins car example, provide robust evidence supporting the effectiveness of our MPC methodology in achieving accurate and stable control across various systems.

4.3 Gas Transportation Pipeline

In the pursuit of robust control systems, leveraging what we understand from 4D extended Dubins car and Cart Pole systems, we examine the error of a gas pipeline control system to establish the efficacy of the methodology. The analysis yields error values from equation 3.26 for the gas pipeline system in the range of 0.009 to 0.027, varying with the selected parameters. This translates approximately to a 98%-95% accuracy.

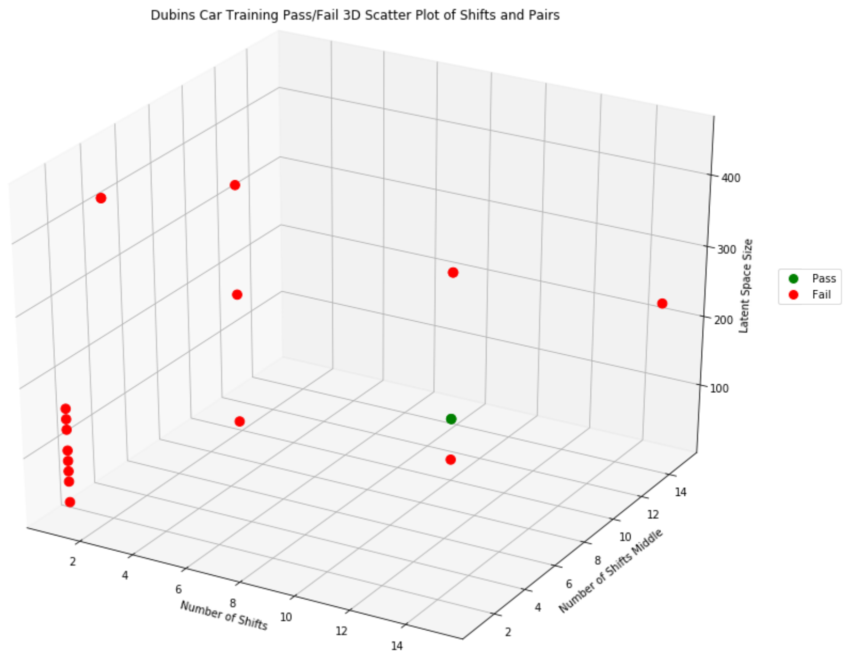

Figure 4.11: 3D plot of Cart Pole Testing runs based of pass/fail criteria under control

The error margins observed for the gas pipeline system are within an acceptable range. Notably, the granular variations in error with respect to shifts in parameter space suggest that the model is responsive to tuning, which is indicative of a well-defined parameter landscape.

The collected data substantiates our confidence in the chosen methodology's potential to precisely engage control mechanisms. As we progress toward the implementation phase, the nuanced understanding of the error dynamics will serve as a compass for further refinement, ultimately enhancing the control system's efficacy.

4.3.1 Post-Training Validation and Hyperparameter Optimization

Integration with Plant Operations

The transition from a simulated environment to real-world application involves close collaboration with the plant operators. The advanced control system will interface with the Supervisory Control and Data Acquisition (SCADA) system, assimilating real-time data to make informed decisions. Operators will input the desired trajectory, aligning with the delivery schedules, into the control system. The system will then generate recommended set points for the various control parameters.

These recommendations are not to be autonomously enacted but rather serve as advisements subject to operator validation. This step is critical to ensure that the suggested

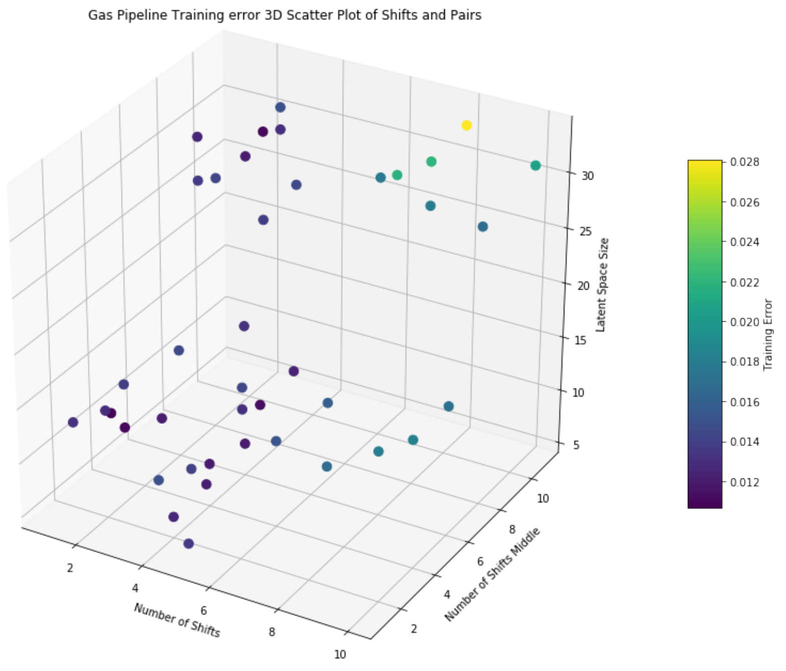

Figure 4.12: 3D plot of Gas Pipeline training runs showcasing Error vs. Hyperparameters

control actions align with the practical and safety considerations inherent in plant operations. The operators' expertise and experience will play a pivotal role in the iterative process of hyperparameter tuning, ensuring that the system's performance is not only theoretically sound but also pragmatically viable.

Iterative Improvement

Through an iterative process, the feedback from the plant operators will contribute to the fine-tuning of the model's hyperparameters. This hands-on approach allows for the accommodation of the plant's unique characteristics and operational contingencies. The objective is to refine the system to a point where it exhibits high reliability and precision, thus providing a reliable decision-support tool for the plant operators.

The ultimate goal of this phase is to harmonize the theoretical model with practical operability, thereby enhancing the efficiency and safety of the gas pipeline system. This phase is not the conclusion but rather an ongoing journey towards perpetual improvement and adaptation to evolving operational demands.

Chapter 5

Conclusion

This study has explored the intricacies of utilizing the concept of Koopman operator within neural networks to perform system identification across diverse dynamical systems such as 4D extended Dubins cars, Cart Pole, and gas transportation pipelines systems. And perform control tasks for 4D extended Dubins cars, and Cart Pole within Linear Model Predictive Control frameworks . Our empirical investigation has provided valuable insights into the efficacy of modelling non-linear systems as linear representation, and the limitations that come when placed under control given the relationship between training errors of the trained neural networks and their performance in control tasks.

5.0.1 Summary of Findings

Our results clearly demonstrate that while minimizing training error is important, it is not the sole indicator of a successful control system. We've identified that a balance between prediction accuracy and control robustness is crucial, particularly when generalizing to new situations. The sweet spot of hyperparameter values was found to be critical for optimal performance, necessitating empirical testing beyond theoretical predictions.

5.0.2 Significance of the Research

The findings underscore the complexity of designing neural networks that not only predict accurately but also control effectively. This is exemplified by the successful application of MPC in the Cart Pole control tasks, where the trained models facilitated the system to reach and maintain specified target states reliably, And in 4D extended Dubins car, where the controller effectively tracks the target trajectory just as well as a NMPC with the true dynamics provided.

5.0.3 Implications for Control System Design

The research suggests that a nuanced understanding of the error dynamics and hyperparameter tuning is essential for the development of robust control systems. The observed consistency across different systems solidifies our methodology as a versatile approach in the face of non-linear dynamics.

5.0.4 Future Work

Future Research

Further research could expand upon the integration of Koopman-based system identification with reinforcement learning algorithms, potentially overcoming the limitations associated with state constraints. Additionally, exploring the adaptability of our methodology to other complex systems could unveil more universal applications.

Validation of Gas Pipeline

The next steps in order to test the viability of our method in a real-world application, we will be validating the performance of the neural networks trained to solve the gas pipeline control problem.

5.0.5 Final Thoughts

In conclusion, the combination of neural networks and MPC holds significant promise for the advancement of control systems in various applications. By bringing the advantages of linear systems in control applications and leveraging the learning capabilities of neural networks. Our methodological approach, while not without its limitations, offers a compelling avenue for efficient and effective system control, striking a balance between the depth of machine learning and the precision of traditional control theory.

Bibliography

[1] I. Abraham and P. J. Goulart. Data-driven koopman operators for model predictive control. *IEEE Control Systems Letters*, 4(2):307–312, 2019.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[3] Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLOS ONE*, 11(2):e0150171, 2016.

[4] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.* Cambridge University Press, 2019.

[5] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLoS ONE*, 11(2):e0150171, 2016.

[6] Carl Folkestad, Daniel Pastor, Igor Mezic, Rainer Mohr, Maria Fonoberova, and Joel Burdick. Extended dynamic mode decomposition with learned koopman eigenfunctions for prediction and control. In *2020 American Control Conference (ACC)*, pages 1712–1718, 2020.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

[8] Thomas Kailath. *Linear Systems.* Prentice-Hall, 1980.

[9] Thomas Kailath. *Linear Systems.* Prentice-Hall Information and System Sciences Series. Prentice-Hall, 1980.

[10] Anatole B. Katok and Boris Hasselblatt. *Introduction to the Modern Theory of Dynamical Systems.* Cambridge University Press, 1997.

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

[12] Milan Korda and Igor Mezić. Convergence rates of moment-based spectral methods for koopman operator approximation and system identification. *arXiv preprint arXiv:1812.01857*, 2018.

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

[14] Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[15] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

[16] David Q. Mayne, James B. Rawlings, Chetan V. Rao, and Peter O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[17] Igor Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41(1):309–325, 2005.

[18] Shahab Mohaghegh. Applications of artificial intelligence & data mining techniques in the petroleum & natural gas industry. *SPE*, 2000.

[19] Phil Muhlbauer. Pipeline rules of thumb handbook. *Elsevier*, 2014.

[20] Pipelinefacts. Canadian pipeline transportation system. *Pipelinefacts*, 2016.

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

[22] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[23] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[24] 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.

Appendix A

Code

`https://github.com/Sriraj-Meenavilli/Koopman_Control.git`