Data-Driven Control with Learned Dynamics

Wenjian Hao
Clemson University, haowjz@gmail.com

Data Driven Control with Learned Dynamics

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

by
Wenjian Hao
August 2020

Acknowledgments

The past two years at Clemson University have been a great journey in my life. During this period, I obtained not only lots of technical knowledge but also a massive help and happy memories from the university community, here I would like to express my sincere gratitude to those who helped me.

First of all, I would like to thank my advisor, Dr. Yiqiang Han, for introducing me to the fields of deep reinforcement learning, dynamical systems, and control. I am very grateful for the opportunity to get involved with many incredible CI projects like autonomous drones and autonomous race cars. I learned a lot of cutting edge knowledge and got much successful experience while doing these great projects. I also appreciate my committee members, Dr. Yue Wang and Dr. Mohammad Naghnaeian, for their guidance and support. Likewise, I would like to thank Professor Umesh Vaidya for his critical input and advice when we worked together to develop a novel algorithm that combines deep neural network and Koopman operator.

My gratitude also goes to my colleagues and friends who offered me help and cooperated for class team projects. It is you who makes my experience unforgettable and precious.

Finally, I would like to express my gratitude to the people dearest to me. I want to thank my parents and family for their love, support, and wisdom throughout my life. Thank you for being part of this journey, even when being at a distance.

Abstract

This research focuses on studying data-driven control with dynamics that are actively learned from machine learning algorithms. With system dynamics being identified using neural networks either explicitly or implicitly, we can apply control following either a model-based approach or a model-free approach. In this thesis, the two different methods are explained in detail and finally compared to shed light on the emerging data-driven control research field.

In the first part of the thesis, we first introduce state-of-art Reinforcement Learning (RL) algorithm representing data-driven control using a model-free learning approach. We discuss the advantages and shortcomings of the current RL algorithms and motivate our study to search for a model-based control which is physics-based and also provides better model interpretability. We then propose a novel data-driven, model-based approach for the optimal control of the dynamical system. The proposed approach relies on the Deep Neural Network (DNN) based learning of Koopman operator and therefore is named as Deep Learning of Koopman Representation for Control (DKRC). In particular, DNN is employed for the data-driven identification of basis function used in the linear lifting of nonlinear control system dynamics. One a linear representation of system dynamics is learned, we can implement classic control algorithms such as iterative Linear Quadratic Regulator (iLQR) and Model Predictive Control (MPC) for optimal control design. The controller synthesis is purely data-driven and does not rely on prior domain knowledge. The OpenAI Gym environment is used for simulations of various control problems. The method is applied to three classic dynamical systems on OpenAI Gym environment to demonstrate the capability.

In the second part, we compare the proposed method with a state-of-art model-free control method based on an actor-critic architecture – Deep Deterministic Policy Gradient (DDPG), which has been proved to be effective in various dynamical systems. Two examples are provided for comparison, i.e., classic Inverted Pendulum and Lunar Lander Continuous Control. We compare these two methods in terms of control strategies and the effectiveness under various initialization conditions from the results of the experiments. We also examine the learned dynamic model from DKRC with the analytical model derived from the Euler-Lagrange Linearization method, demonstrating the accuracy in the learned model for unknown dynamics from a data-driven sample-efficient approach.

Key Words: Dynamical System and Control, Koopman Operator, Reinforcement Learning, Deep Deterministic Policy Gradient (DDPG), Model Predictive Control (MPC), Deep Neural Network (DNN)

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The data-driven control of dynamical systems has been an emerging research area with various applications in robotics, manufacturing systems, autonomous vehicles, and transportation networks. There is a rising interest in shifting from completely model-based to data-driven control paradigm with the increasing complexity of engineered systems and easy access to a large amount of sensor data. One successful example is the Deep Reinforcement Learning (DRL), which consists of deploying Deep Neural Networks (DNN) to learn optimal control policy. It is arising as one of the powerful tools for data-driven control of dynamic systems [37]. However, the downside of only relying on a neural network approach is also obvious. The use of DNN is often being called a "black-box" approach. The problem of control design for a system with nonlinear dynamics is recognized to be a particularly challenging problem. More recently, the use of linear operators from the ergodic theory of dynamical systems has offered some promises towards providing a systematic approach for nonlinear control [4,16,20,24,26,28,30,33,34,39,40].

The controller synthesis for dynamic systems in the model-based and model-free setting has a long history in control theory literature. However, some challenges need to be overcome for the successful extension of linear operator-based analysis methods to address the synthesis problem. The basic idea behind the linear operator framework involving Koopman and Perron-Frobenius operator is to lift the finite-dimensional nonlinear dynamics to infinite-dimensional linear dynamics. The finite-dimensional approximation of the linear operator is obtained using the time series data of the dynamic system.

One of the main challenges is the selection of appropriate choice of finite basis functions used in the finite but high dimensional linear lifting of nonlinear dynamics. The standard procedure is to make a prior choice of basis function such as radial basis function or polynomial function for the finite approximation of the Koopman operator. However, such an approach often does not take advantage of the known physics or the data in choosing the basis function.

An alternate approach based on the power of Deep Neural Network (DNN) could be used for the data-driven identification of the appropriate basis function in the autonomous (uncontrolled) setting [21, 47]. A dummy representation of the Deep Neural Network (DNN) structure is shown in Figure 1.1.

The conjecture is that physics is reflected in the data, and DNNs are efficient in exploiting the power of the data. In this work, we extend the application of DNN for the selection of basis functions in a controlled dynamical system setting.

The main contributions of our method are as follows. We extend the use of DNN for the finite-dimensional representation of the Koopman operator in a controlled dynamical system setting. Our data-driven learning algorithm can automatically search for a rich set of suitable basis functions to construct the approximated linear model in the lifted space. The OpenAI Gym environment is employed for data generation and training of DNN to learn the basis functions [3]. Example systems from OpenAI Gym environment employed for the design of reinforcement learning

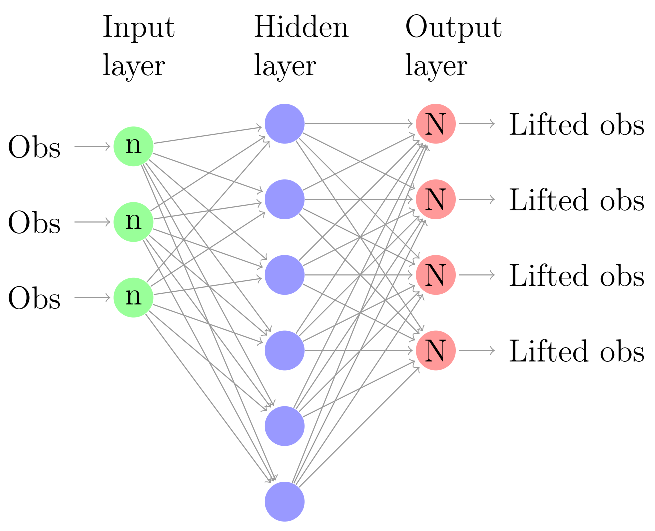


Figure 1.1: Simple neural network structure example. n: The dimension of the input observations N: The dimension of lifted observations

control is used to demonstrate the capability of the developed framework. This work is on the frontier to bridge the understanding of model-based control and model-free control (e.g. Reinforcement Learning) from other domains. The proposed comparison framework between the two approaches is depicted in Figure 1.2.

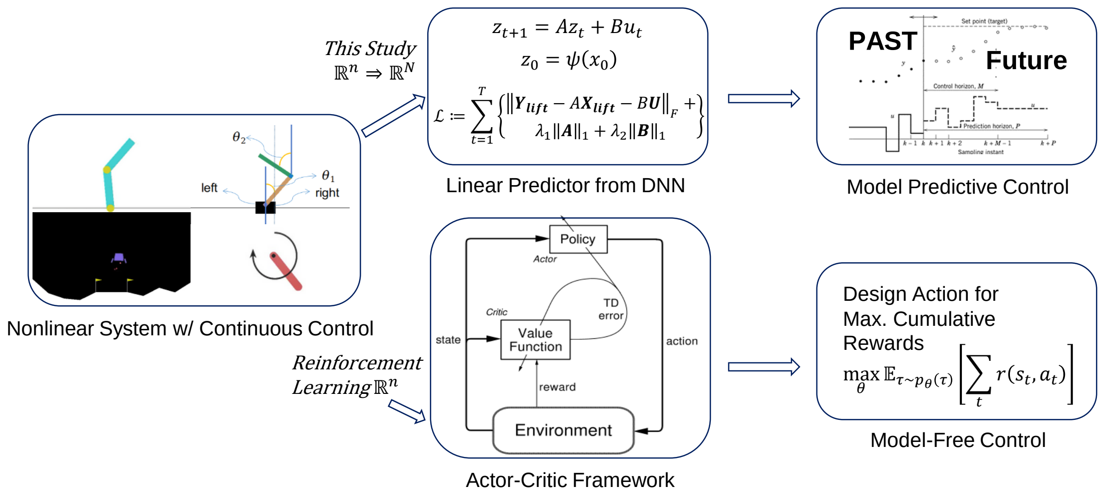

This Study
$\mathbb{R}^n \Rightarrow \mathbb{R}^N$
$z_{t+1} = Az_t + Bu_t$
$z_0 = \psi(x_0)$
$\mathcal{L} := \sum_{t=1}^{T} \left\{ \begin{matrix} \|\boldsymbol{Y_{lift}} - A\boldsymbol{X_{lift}} - B\boldsymbol{U}\|_F + \\ \lambda_1 \|\boldsymbol{A}\|_1 + \lambda_2 \|\boldsymbol{B}\|_1 \end{matrix} \right\}$
Linear Predictor from DNN
PAST
Future
Set point (target)
Control horizon, M
Prediction horizon, P
Sampling instant
Model Predictive Control
θ_2
θ_1
left
right
Nonlinear System w/ Continuous Control
Policy
Actor
Critic
TD error
Value Function
state
action
reward
Environment
Actor-Critic Framework
Reinforcement Learning $\mathbb{R}^n$
Design Action for Max. Cumulative Rewards
$\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$
Model-Free Control

Figure 1.2: Deep Koopman Representation for Control (DKRC) framework with comparison to Reinforcement Learning

1.1 Koopman Operator

The following sections introduce several key concepts of algorithms and the state-of-art research progress in those areas.

Koopman operator theory is an alternative formalism for dynamical systems theory, which offers excellent utility in the analysis and control of nonlinear and high-dimensional systems using data [25]. Koopman operator is an infinite-dimensional linear operator that can transfer a nonlinear system to a high-dimensional linear system. It fully captures all properties of the underlying dynamical system provided that the space of observables $f(x_t)$ contains the components of the states x_t [20]. Considering an uncontrolled dynamic system in Equation 1.1.

$$x_t = f(x_t) \tag{1.1}$$

The Koopman operator K would be used to transform the representation of the system in the following format as shown in Equation 1.2. The detailed use of this linear representation will be introduced in later sections together with the proposed model-based control design.

$$(K\psi) = \psi(F(x)) \tag{1.2}$$

A simple explanation diagram of Koopman Operator is exhibited in Figure 1.3. As shown in the figure, the K is the koopman operator, ψ represents an infinite dimensional basis function, $F(x_t)$ is the state evolution operator, $\psi(F(x_t))$ is the linear lifted observation space, n is the dimension of the observation, m is the dimension of the lifted observation, Therefore, Koopman operator is defined as the composition of ψ with F. Common choices for the basis function include Hermite polynomials and radial basis functions [44].

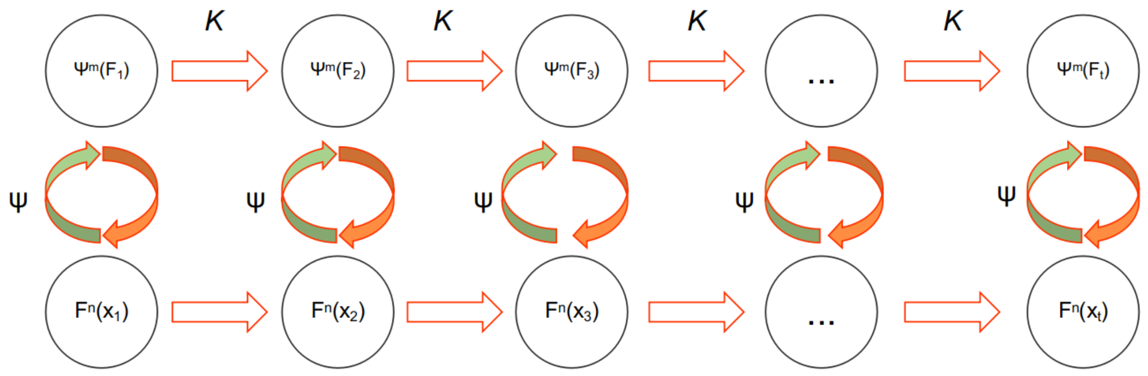


Figure 1.3: Koopman Operator Scheme

1.2 Model Predictive Control

Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. Model predictive controllers rely on dynamic models of the process, most often linear empirical models obtained by system identification. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly, thus differing from Linear-Quadratic Regulator (LQR). Also MPC has the ability to anticipate future events and can take control actions accordingly. PID controllers do not have this predictive ability. MPC is nearly universally implemented as a digital control, although there is research into achieving faster response times with specially designed analog circuitry [42].

In this study, we implement the Model Predictive Control (MPC) after learning the dynamics of the lifted high-dimensional linear system with DKRC. Other methods, such as regression or DAGGER (i-LQR or MCTS for planning) techniques have been introduced after the learned model is available. However, those methods suffer from distribution mismatch problems or issues with the performance of open-loop control in stochastic domains. We propose using MPC iteratively, which can provide robustness to small model errors that benefit from the close loop control. As mentioned by authors in Ref [6], if the system dynamics were to be linear and the cost function convex, the global optimization problem solved by MPC methods would be convex (To solve the convex problem, we used CVXPY [8] in this study.). Under this assumption, the convex programming optimization techniques could be promising to achieve some theoretical guarantees of convergence to an optimal solution. Therefore MPC would undoubtedly perform better in a linear setting. MPC is an online optimization solver method that seeks optimal state solutions at each time step under certain defined constraints. Therefore it features closed-loop planning, which exhibits higher robustness than other simpler models. It has been proven to be tolerant of the modeling errors within Adaptive Cruise Control(ACC) simulation [10] [49]. MPC represents the state-of-the-art for the practice of real-time optimal control [22].

1.3 Reinforcement Learning

To compare the advantages and disadvantages of the proposed model-based method, we examine the proposed method with another model-free control method (Deep Deterministic Policy Gradient) in the second part of this study. Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm based on an actor-critic framework. The DDPG and its variants have been demonstrated to be very successful in designing optimal continuous control for many dynamical systems. However, compared to the above model-based optimal control, DDPG employs a more black-box style neural network structure that outputs control directly based on learned DNN parameters. The DDPG and its variants typically require a significant amount of training data and a good design of the reward function to achieve good performance without learning an explicit model of the system dynamics. We use this section to introduce several of the key concepts to better understand this state-of-the-art reinforcement learning algorithm.

1.3.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy algorithm, which uses DNN function approximators to learn policies in high-dimensional, continuous action spaces [38]. DDPG is a combination of the deterministic policy gradient approach and insights from the success of Deep Q Network (DQN) [27]. DQN is a policy optimization method that achieves optimal policy by inputting observation and updates policy by back-propagate policy gradients. However, DQN can only handle discrete and low-dimensional action spaces, limiting its application considering many control problems have high-dimensional observation space and continuous control requirements. The innovation of DDPG is that it extends the DQN to continuous control domain and higher dimensional system and has been claimed that it robustly solves more than 20 simulated physics tasks, including classic problems such as cart-pole swing-up, dexterous manipulation, legged locomotion and car driving [38]. In addition, DDPG has been demonstrated to be sample-efficient compared to the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is also a black-box optimization method widely used for robot learning [1].

A schematic diagram of the DDPG framework is shown in Figure 1.4.

The control goal of the DDPG algorithm is to find control action u to maximize the rewards (Q-values) evaluated at the current time step. The training goal of the algorithm is to learn neural network parameters for the above networks so that rewards

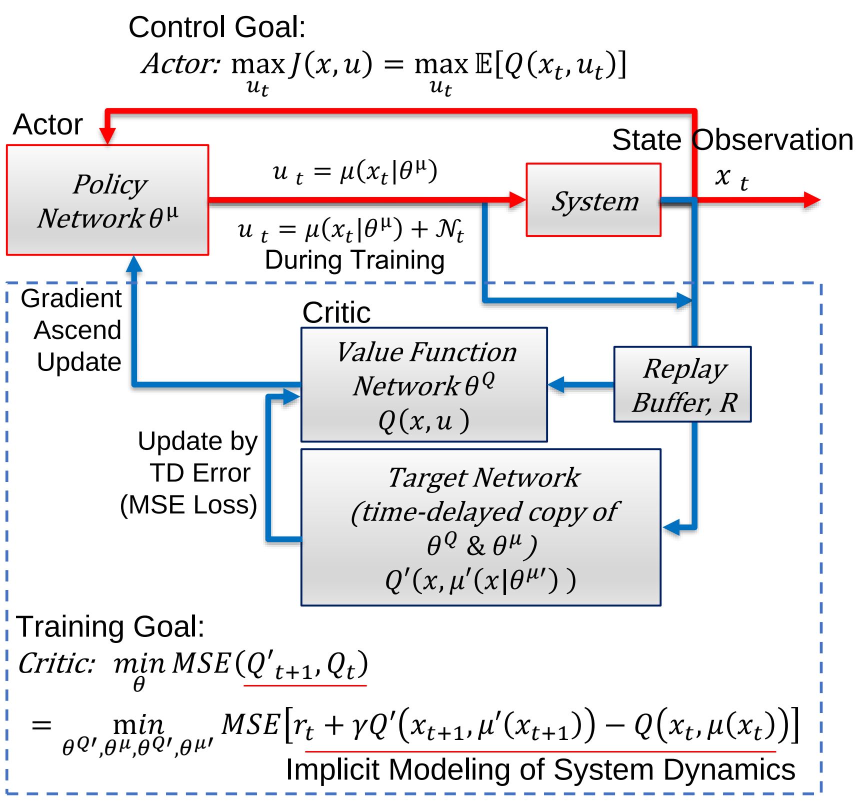

Control Goal:
Actor: $\max_{u_t} J(x,u) = \max_{u_t} \mathbb{E}[Q(x_t,u_t)]$
Actor
Policy Network θ^μ
$u_t = \mu(x_t|\theta^\mu)$
$u_t = \mu(x_t|\theta^\mu) + \mathcal{N}_t$
During Training
System
State Observation
x_t
Gradient Ascend Update
Critic
Value Function Network θ^Q $Q(x,u)$
Replay Buffer, R
Update by TD Error (MSE Loss)
Target Network (time-delayed copy of θ^Q & θ^μ) $Q'(x,\mu'(x|\theta^{\mu'}))$
Training Goal:
Critic: $\min_\theta MSE(Q'_{t+1}, Q_t)$
$= \min_{\theta^{Q'},\theta^\mu,\theta^{Q'},\theta^{\mu'}} MSE[r_t + \gamma Q'(x_{t+1},\mu'(x_{t+1})) - Q(x_t,\mu(x_t))]$
Implicit Modeling of System Dynamics

Figure 1.4: Schematics of DDPG framework

can be maximized at each sampled batch, while still being deterministic. The training of the DDPG network will then follow a gradient ascend update from each iteration, as defined in Equation 1.3

$$\nabla_{\theta^\mu} J \approx \mathbb{E}[\nabla_{\theta^\mu} Q(x,u|\theta^Q)|x = x_t, u = \mu(x_t|\theta^\mu)] \tag{1.3}$$

In Equation 1.3, we have two separate networks involved: Value function network θ^Q and policy network θ^μ. To approximate the gradient from those two networks, we design the training process relying on one replay buffer and four neural networks: Actor, Critic, Target Actor, Target Critic.

At the gathering training samples stage, we use a replay buffer to store samples. A replay buffer of DDPG essentially is a buffered list storing a stack of training samples (x_t, u_t, r_t, x_{t+1}). During training, samples can be drawn from the replay buffer in batches, which enables training using a batch normalization method [5]. For each stored sample vector, t is current time step, x is the collection of state observations at current time step t, r is numerical value of reward at current state, u is the action of time step t, and finally, the x_{t+1} is the state in the next time step $t+1$ as a result of taking action u_t at state x_t. Replay buffer makes sure an independent and efficient sampling [7].

As discussed above, the DDPG framework is a model-free reinforcement learning architecture, which means it does not seek an explicit model for the dynamical system. Instead, we use a value function network, also called Critic network $Q(s, a|\theta^Q)$, to approximate the result from a state-action pair. The result of a trained Critic network estimates the expected future reward by taking action u_t at state x_t. If we take the gradient of the change in the updated reward, we will be able to use that gradient to update our Policy network, also called Actor network. At the end of the training, we obtain an Actor network capable of designing optimal control based on past experience. Since there are two components in the formulation of the gradient of reward in Equation 1.3, we can apply the chain rule to break the process into evaluating gradients from Actor and Critic networks separately. The total gradient is then defined as the mean of the gradient over the sampled mini-batch, where N is the size of the training mini-batch taken from replay buffer R, as shown in Equation 1.4. Silver et al. showed proof in Ref. [31] that Equation 1.4 is the policy gradient that can guide the DDPG model to search a policy network to yield the maximum expected reward.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_t [\nabla_u Q(x, u|\theta^Q)|_{x=x_t, u=\mu(x_t)} \nabla_{\theta^\mu} \mu(x|\theta^\mu)|_{x_t}] \tag{1.4}$$

In this case, dynamical responses from the system are built into the modeling of the Critic network, thus indirectly modeled. The accuracy in predicted reward values is used as the only training criterion in this process, signifying the major difference between the model-free reinforcement learning and the model-based optimal control method such as DKRC. Some of the later observations in different behaviors from model-based vs. model-free comparison have roots stemming from this fundamental difference.

We separate the training of the DDPG into the training of the Actor and Critic networks.

Actor, or policy network (θ^μ), is a simple neural network with weights θ^μ, which takes states as input and outputs control based on a trained policy $\mu(x, u|\theta^\mu)$. The Actor is updated using the sampled policy gradient generated from the Critic network, as proposed in Ref. [32].

Critic, or value function network, is a neural network with weights θ^Q, which takes a state-action pair (x_t, u_t) as input. We define a temporal difference error term, e_t (TD error), to track the error between the current output compared to a target Critic network with future reward value using future state-action pair as input in the next time step. The total loss from the Critic network during a mini-batch is defined as L in Equation 1.5, which is based on Q-learning, a widely used off-policy algorithm [41].

$$
\begin{aligned}
e_t &= (r_t + \gamma Q^{'}(x_{t+1}, \mu^{'}(x_{t+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})) - Q(x_t, u_t|\theta^Q) \\
L &= \frac{1}{N}\sum_t {e_t}^2
\end{aligned}
\tag{1.5}
$$

In this Critic loss function in Equation 1.5, the terms with prime symbol ($'$) is the Actor/Critic model from the target networks. Target Actor $\mu^{'}$ and target Critic $Q^{'}$ of DDPG are used to sustain a stable computation during the training process. These two target networks are time-delayed copies of the actual Actor/Critic networks in training, which only take a small portion of the new information from the current iteration to update target networks. The update scheme of these two target networks are defined in Equation 1.6, where τ is Target Network Hyper Parameters Update rate super parameter with a small value ($\tau \ll 1$) to improve the stability of learning.

$$
\begin{aligned}
\theta^{Q^{'}} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q^{'}} \\
\theta^{\mu^{'}} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu^{'}}
\end{aligned}
\tag{1.6}
$$

To train DDPG, we initialize four networks $(Q(s, a|\theta^Q), \mu(s|\theta^\mu), Q^{'}(s, a|\theta^{Q^{'}}), \mu^{'}(s|\theta^{\mu^{'}}))$ and a list buffer (R) first, then select an initial action from actor, executing action a_t and observe reward r_t and new state s_{t+1}, store (s_t, a_t, r_t, s_{t+1}) in the buffer R, sampling mini training batch input randomly from R, updating critic by Equation 1.5, updating actor by Equation 1.4, updating target actor and critic by Equation 1.6 until loss value converges.

1.4 Thesis Organization

The work will be presented in the following order: Ch 2 introduces the proposed problem setup; Ch 3 formulates the proposed algorithm; Ch 4 details the results and discussions from the deployment of proposed algorithm; In Ch 5, since DKRC and DDPG are both capable of solving dynamic systems control problems with high-dimensional and continuous control space, direct comparison is desirable by constructing the two algorithms from the ground up and applying them to solve the same tasks. In this chapter, we also compare the DKRC to the analytical model obtained by the classic Euler-Lagrange Linearization method. The comparison are examined in the Inverted pendulum environment and Lunar Lander Continuous Control environment in OpenAI Gym [3]. Ch 6 concludes the thesis.

Chapter 2

Research Problem Setup

This chapter consists of the processes we have taken to develop the algorithm. We generally attempt three different ways to solve the proposed research problem, two of them failed, and only one succeeded in the end.

2.1 Problem Set-up

Consider a discrete-time, nonlinear dynamical system along with a set of observation data of length T:

$$\begin{gathered} x_{t+1} = F(x_t, u_t) \\ \boldsymbol{X} = [x_1, x_2, \dots, x_T], \quad \boldsymbol{Y} = [y_1, y_2, ..., y_T], \quad \boldsymbol{U} = [u_1, u_2, \dots, u_T] \end{gathered} \tag{2.1}$$

where, $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^m$ is the state and control input respectively, and $y_t = x_{t+1}$ for $t = 1, \dots, K$. The objective is to provide a linear lifting of the nonlinear dynamical system for the purpose of control. The Koopman operator theory can be used to lift the nonlinear control dynamics to linear control dynamics. The Koopman operator ($\mathcal{K}$) is an infinite-dimensional linear operator defined on the space of functions as follows:

$$[\mathcal{K}g](x) = g \circ F(x,u) \tag{2.2}$$

For more details on the theory of Koopman operator in a dynamical system and control setting, refer to Ref. [19, 43, 45, 46]. Related work on extending Koopman operator methods to controlled dynamical systems can be found in References [2, 16, 17, 20, 23, 29, 48]. While the Koopman based lifting for the autonomous system is well understood and unique, the dynamical system can be lifted in different ways. In particular, for the control of affine system, the lifting in the functional space will be bi-linear [15, 23]. Given the computational focus of this paper, we resort to the linear lifting of the control system. The linear lifting also has the advantage of using state of the art control methodologies such as linear quadratic regulator (LQR) and Model Predictive Control (MPC) from linear system theory. The linear lifting of nonlinear control system is given by

$$\Psi(x_{t+1}) = A\Psi(x_t) + Bu_t \tag{2.3}$$

where $\Psi(x) = [\psi_1(x), \ldots, \psi_N(x)]^\top \in \mathbb{R}^{N>>n}$ is the state or the basis in the lifted space. The matrix $A \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times m}$ are the finite dimensional approximation of the Koopman operator for control system in the lifted space. One of the main challenges in the linear lifting of the nonlinear dynamics is the proper choice of basis functions used for lifting. While the use of radial basis function, polynomials, and kernel functions is most common, the choice of appropriate basis function is still open. Lack of systematic approach for the selection of appropriate basis has motivated the use of Deep Neural Network (DNN) for data-driven learning of basis function. The use of DNN has been explored for the learning of basis functions in uncontrolled settings.

The planned control inputs considered in this study are in continuous space and later will be compared to state-of-art model-free learning methods such as Reinforcement Learning. An overview of the scope of this work is shown in Figure 1.2. Both the two methods (our Deep Koopman Representation for Control (DKRC) approach and Reinforcement Learning) apply a data-driven model-free learning approach to learn the dynamical system. However, our approach seeks a linear representation of nonlinear dynamics and then design the control using a model-based approach, which will be discussed in later sections.

2.2 Proposed Solutions

In this section, we demonstrate how we develop the algorithm, and the processes are divided into three attempts.

2.2.1 The First Solution

In order to solve the problem defined in Equation 2.3. Firstly, We attempt to use three neural networks $(\psi(x_{t+1}|\theta), A(\psi(x_t|\theta)|\theta^A), B(u_t|\theta^B))$ to get a solution for Equation 2.3, which transfers the loss function of the training process to Equation 2.4.

$$loss_{attempt1} = ||\psi(x_{t+1}|\theta) - A(\psi(x_t|\theta)|\theta^A) - B(u_t|\theta^B)||_2 \tag{2.4}$$

Where, $A(\psi(x_t|\theta)|\theta^A)$ takes the output of $\psi(x_{t+1}|\theta)$ as input state and outputs the same dimensional lifted state as $\psi(x_{t+1}|\theta)$. A scheme of the attempt is shown in Figure 2.1.

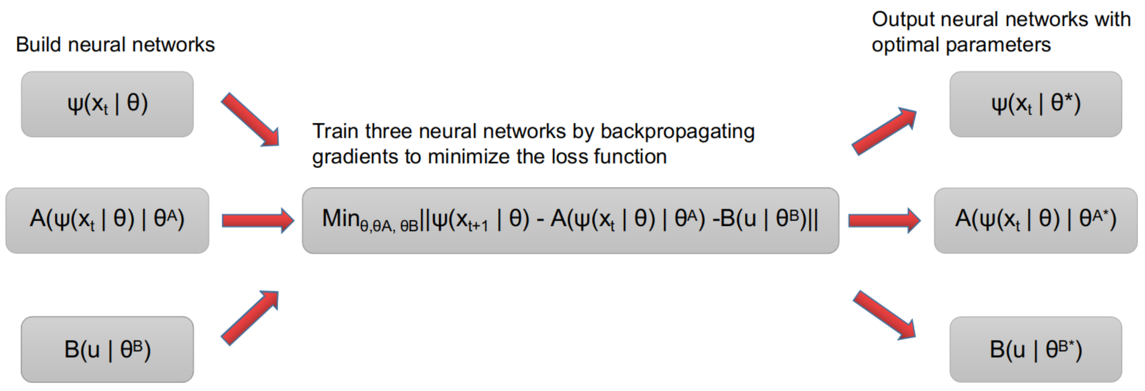

Build neural networks
ψ(x_t | θ)
A(ψ(x_t | θ) | θ^A)
B(u | θ^B)
Train three neural networks by backpropagating gradients to minimize the loss function
Min_{θ,θA, θB}||ψ(x_{t+1} | θ) - A(ψ(x_t | θ) | θ^A) -B(u | θ^B)||
Output neural networks with optimal parameters
ψ(x_t | θ*)
A(ψ(x_t | θ) | θ^{A*})
B(u | θ^{B*})

Figure 2.1: The first failure solution flow chart

The problem of this attempt is that: (1) To get A, B matrices, we can not add any layer and bias in neural networks $A(\psi(x_t|\theta)|\theta^A)$ and $B(u_t|\theta^B)$, which leads to an imprecise result. (2) Besides, we find that when the neural networks are trained to minimize the loss function using gradients back-propagation, it tends to simply push all three neural networks' outputs close to zero, which is one of the reasons why the solution only takes trivial control no matter what states it accepted during result test.

2.2.2 The Second Solution

In this solution, we only use one neural network $\psi(x_{t+1}|\theta)$ to solve Equation 2.3. At training epoch n, loss function is defined in Equation 2.5.

$$loss_{attempt2} = ||\psi(x_{t+1}|\theta) - A_n \times \psi(x_t|\theta) - B_n \times u_t||_2 \tag{2.5}$$

Where, A_n, B_n is defined in Equation 2.6 and A_1, B_1 is randomly generated at the first training epoch.

$$[A_n, B_n] = \frac{1}{L}\sum_{t=1}^{L} \psi_{t+1} \begin{bmatrix} \psi_t \\ u_t \end{bmatrix} \left(\begin{bmatrix} \psi_t & u_t \end{bmatrix} \begin{bmatrix} \psi_t \\ u_t \end{bmatrix} \right)^{\dagger} \tag{2.6}$$

where, L is the number of training samples, † denotes psedoinverse.

When $n > 1$, A, B is updated as following:

$$A_n = \epsilon A_n + (1-\epsilon)A_{n-1},\ \ B_n = \epsilon B_n + (1-\epsilon)B_{n-1},\ \ \epsilon \ll 1$$

ϵ is selected to get a smooth parameter update. A general solution flow chart is exhibited in Figure 2.2.

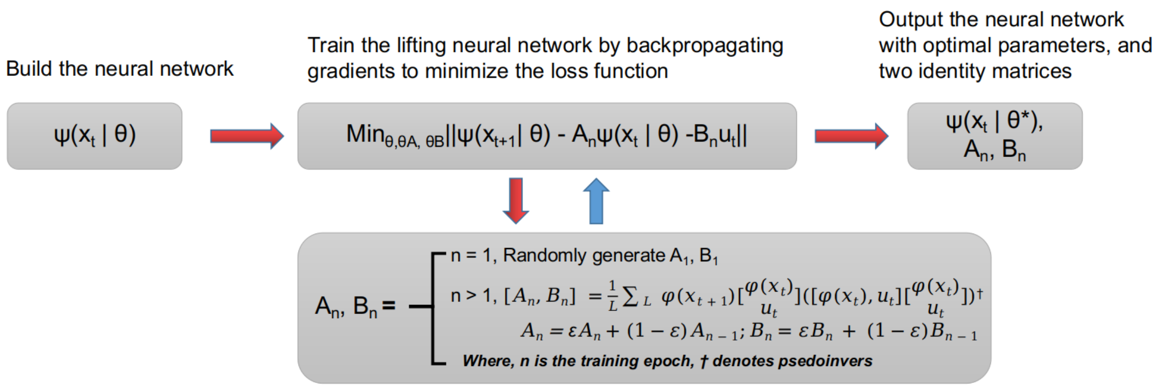

Build the neural network
Train the lifting neural network by backpropagating gradients to minimize the loss function
Output the neural network with optimal parameters, and two identity matrices
ψ(x_t | θ)
Min_{θ,θA, θB}||ψ(x_{t+1}| θ) - A_nψ(x_t | θ) -B_nu_t||
ψ(x_t | θ*), A_n, B_n
A_n, B_n =
n = 1, Randomly generate A_1, B_1
n > 1, [A_n, B_n] = 1/L Σ_L φ(x_{t+1})[φ(x_t); u_t]([φ(x_t), u_t][φ(x_t); u_t])†
A_n = εA_n + (1 − ε)A_{n−1}; B_n = εB_n + (1 − ε)B_{n−1}
Where, n is the training epoch, † denotes psedoinvers

Figure 2.2: The Second failure solution flow chart

The problem of this solution is that: (1) The loss function will converge after the first epoch as a result of that the A, B matrices are obtained mathematically; (2) Besides, the tiny loss value is also a challenging work for neural networks to update its parameters, which means it is hard to converge after the first training epoch. The result of this solution is that it cannot take reasonable control and fails at each game.

2.3 Lessons Learned

After the above two failed attempts, we decided to use only one neural network to represent all the training terms and only update the A, B matrices after the entire training process converges. We also integrate another loss function to ensure the lifted high-dimensional linear system controllable, which is concluded in the next chapter - Algorithm Proposed.

Chapter 3

Algorithm Proposed

This chapter exhibits details of the data-driven control algorithm we develop and also shows the eigenfunctions gotten by the algorithm in the swing-up pendulum system with/without control.

3.1 Deep Learning of Koopman Representation for Control

To enable deep Koopman representation for control, the first step is to use the DNN approach to approximate Koopman operator for control purposes. Our method seeks a multi-layer deep neural network to automate the process of finding the basis function for the linear lifting of nonlinear control dynamics via Koopman operator theory. Each feed-forward path within this neural network can be regarded as a nonlinear basis function ($\psi_i : \mathbb{R}^n \to \mathbb{R}$), whereas the DNN as a collection of basis functions can be written as $\Psi : \mathbb{R}^n \to \mathbb{R}^N$. Each output neuron is considered as a compounded result from all input neurons. Therefore, the mapping ensures that our Koopman operator maps functions of state space to functions of state space, not states to states.

Unlike existing methods used for the approximation of the Koopman operator, where the basis functions are assumed to be known apriori, the optimization problem used in the approximation of Koopman operator using DNN is non-convex. The non-convexity arises due to simultaneous identification of the basis functions and the linear system matrices. The setup of our DNN training process is listed below to address this concern.

During the training of DNN, the observables in state-space (x) are segmented into data pairs to guide the automated training process, whereas the output from the DNN is the representation of the lifting basis function for all data pairs ($\psi(x)$). This method is completely data-driven, and basis functions are learned directly from training datasets.

The input dataset is split into three sets, whereas 70% of the data are used as training samples, 15% as validation, and another 15% as testing cases. At each layer in between the input and output layers, we apply hyperbolic-tangent (tanh) non-linear activation functions with biases at each neuron. Other activation functions choices, e.g., ELU, RELU, Leaky RELU, have also been widely used in other deep learning application for fast convergence. We picked hyperbolic-tangent function to preserve non-linear behavior in the network. Additional machine learning techniques, such as dropout and layer normalization are also applied to prevent overfitting. Dropout is an extra layer in the neural network to randomly select a portion of the neuron to skip updating in this iteration. A typical range of dropout value is between 0.5 and 0.8. Layer Normalization technique is employed to ensure that the training process not be dominated by a specific field in the input data. ADAM [18] and ADAGRAD [9] algorithms are used as DNN training optimizers. These optimizers are used at the backend to handle the adaptive learning rate, how to take gradients and update network weights, etc. All training processes are done using Pytorch 1.4 platform with NVIDIA V100 GPUs on an NVIDIA DGX-2 GPU supercomputer cluster. The deployment of the code has been tested on personal computers with NVIDIA GTX 10-series GPU cards. During training, the DNN will iterate through epochs to minimize a loss term, where differences between two elements in lifted data pairs will be recorded and minimized in batch. At the convergence, we obtain a DNN, which automatically finds optimal lifting functions to ensure both the Koopman transformation (Equation 2.1) and linearity (Equation 2.3) by tuning the DNN's loss function to a minimum. A schematic illustration of the Deep Koopman Representation learning can be found in Figure 3.1.

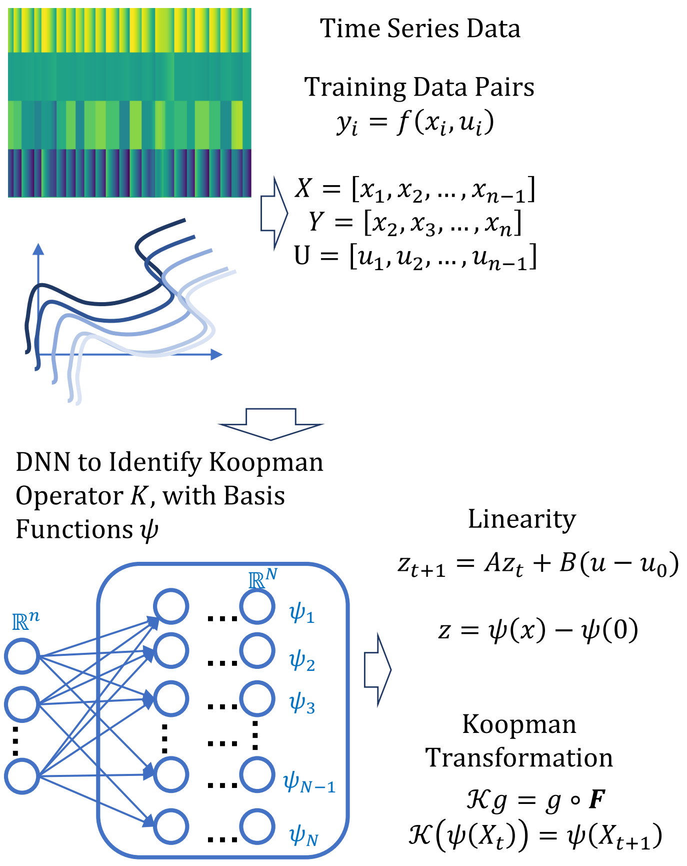

Time Series Data
Training Data Pairs
$y_i = f(x_i, u_i)$
$X = [x_1, x_2, \dots, x_{n-1}]$
$Y = [x_2, x_3, \dots, x_n]$
$U = [u_1, u_2, \dots, u_{n-1}]$
DNN to Identify Koopman Operator K, with Basis Functions ψ
$\mathbb{R}^n$
$\mathbb{R}^N$
ψ_1
ψ_2
ψ_3
ψ_{N-1}
ψ_N
Linearity
$z_{t+1} = Az_t + B(u - u_0)$
$z = \psi(x) - \psi(0)$
Koopman Transformation
$\mathcal{K}g = g \circ \boldsymbol{F}$
$\mathcal{K}(\psi(X_t)) = \psi(X_{t+1})$

Figure 3.1: Koopman Operator Learning using Neural Network Scheme

Once a linear system approximation is obtained though the Deep Koopman Neural Network, we have several state-of-the-art tools to design optimal control. However, the vanilla-flavored optimization processes will attempt to regulate the loss function to a minimum value in lifted space, which may not correspond to the goal position in the non-lifted space. Therefore, it is necessary to account for the effect of constant shift while using the lifting method.

The mapping from state-space to higher-dimensional space requires a modification to the existing control design governing equation, which is equivalent to regulation around a non-zero fixed point for non-linear systems. The affine transformation can be introduced without changing A and B matrices, as shown in Equation 3.1 below:

$$\begin{gathered} \boldsymbol{z} := \Psi(\boldsymbol{x}) - \Psi_0 \\ \boldsymbol{v} := \boldsymbol{u} - u_0 \end{gathered} \tag{3.1}$$

whereas, Ψ_0 is defined as: $\Psi_0 = \Psi(x = 0)$. To approximate u_0, it requires one additional auxiliary network for this constant. The network hyperparameters will then be trained together with the bigger DNN.

After applying these changes, the Equation 2.3 becomes the final expression of the high-dimensional linear model of the system in lifted space:

$$\mathbf{z}_{t+1} = A\mathbf{z}_t + B\mathbf{v}_t \tag{3.2}$$

Correspondingly, the problem we are solving becomes:

$$\min_{A,B,u_0} \sum_k \parallel \mathbf{z}_{t+1} - A\mathbf{z}_t - B(\mathbf{u}) - \mathbf{u_0} \parallel_F \tag{3.3}$$

In particular, the following loss function, $\mathcal{L}$ in Equation 3.4, is used in the training of DNN.

$$\mathcal{L} := \sum_{t=1}^{T} \parallel \mathbf{z}_{t+1} - A\mathbf{z}_t - B(\mathbf{u} - \mathbf{u_0}) \parallel_F \tag{3.4}$$

The problem we are dealing with is non-convex. There is no one-shot solution to solve for multiple unknowns in the problem. We propose the following way to solve for the lifting function, with assumed A and B values at the beginning. We start the training with randomly initialized $A \in \mathbb{R}^{N \times N}$ and $B \in \mathbb{R}^{N \times m}$. The ψ functions are learned within the DNN training, whereas the matrices A and B are kept frozen during each training epoch.

At the end of each epoch, the A and B can be updated analytically by adding new information (with a learning rate of 0.5) based on the Equation 3.5 as follows:

$$[A, B] = \mathbf{z}_{t+1} \begin{bmatrix} \mathbf{z}_t \\ U \end{bmatrix} \left(\begin{bmatrix} \mathbf{z}_t \; U \end{bmatrix} \begin{bmatrix} \mathbf{z}_t \\ U \end{bmatrix} \right)^{\dagger} \tag{3.5}$$

where † denotes psedoinverse. In addition to finding system A and B matrix, following optimization problem is solved to compute the C matrix mapping states from lifted space back to state space.

$$\begin{aligned} &\min_{C} \sum_{t} \parallel x_t - C\Psi(x_t) \parallel_F \\ &\text{s.t.} \quad C\Psi_0 = 0 \end{aligned} \tag{3.6}$$

3.2 Koopman-based Control

Having identified the linear control system in the lifted space along with the basis function using DNN, we proceed with the spectral analysis of the finite dimensional approximation. In particular, the eigenfunctions of the Koopman operator are approximated using the identified A matrix. The eigenfunctions learned from the neural network can be viewed as intrinsic coordinates for the lifted space and provide insights into the transformed dynamics in the higher dimensional space. An example to illustrate this can be seen in Figure 3.2 and Figure 3.3. These two figures are both 3d plots for eigenfunctions obtained from DNN. The difference is that Figure 3.2 represents a single unforced frictionless pendulum, whereas Figure 3.3 denotes the eigenfunctions obtained for single pendulum under control. As can be seen, the first dominant eigenfunctions from both the two figures exhibit a very similar double-pole structure. The uncovered intrinsic coordinates exhibit a trend that is related to the energy levels within the system as base physical coordinates change, i.e., $\dot{\theta}$ is positively related to the kinetic energy, whereas the θ is proportional to the potential energy. Towards the center of the 3d plot, both kinetic energy and potential energy exhibit declined absolute magnitudes. In the controlled case, torque at the joint is being added into the system's total energy count, which seeks a path to bring the energy of the system to the maximum in potential energy and lowest absolute value in kinetic energy. Due to the kinetic energy part ($\frac{1}{2}\dot{\theta}^2$) is dominating the numerical value of the energy, this effect has been reflected in the eigenfunction plot corresponding to the 7th and 8th eigenvalue. The two figures show the feasibility of using eigenfunctions from learned Koopman operator to gain insight of the dynamical system, especially for the control purpose.

Then we can proceed to solve discrete LQR problem for the linear system with a quadratic cost:

$$J(V) = \sum_t z_t^\top C^\top Q C z_t + v_t^\top R v_t \tag{3.7}$$

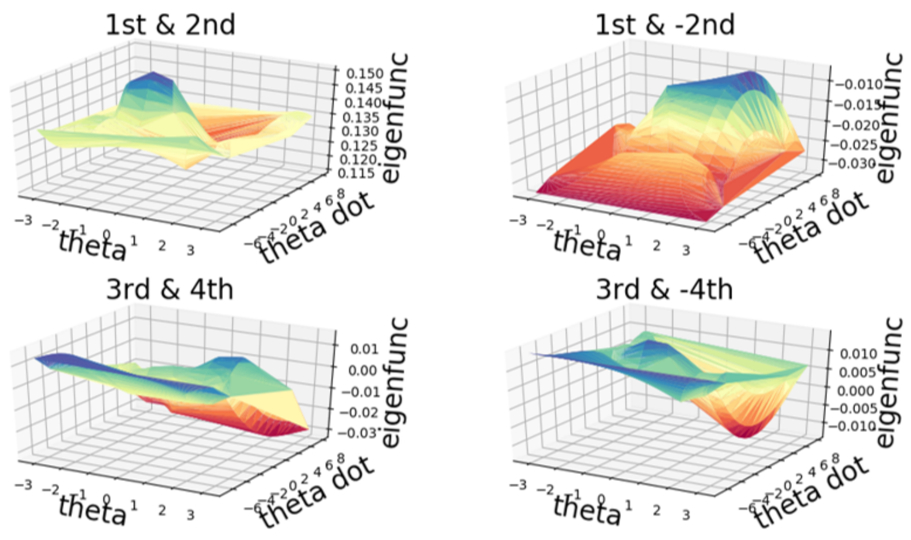


Figure 3.2: Eigenfunctions learned from uncontrolled case, Left: real part of eigenfunctions, Right: imaginary part of the eigenfunctions. The dominant eigenfunction has zero value imaginary part.

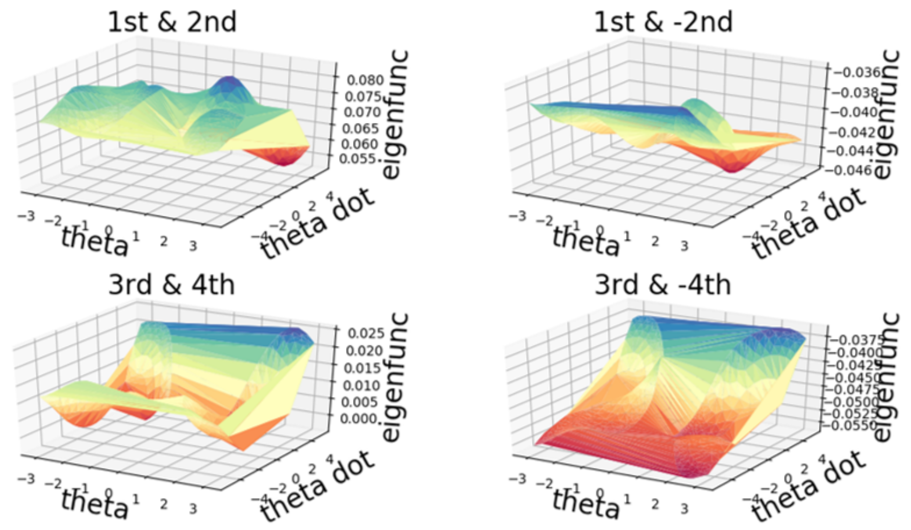


Figure 3.3: Eigenfunctions learned from controlled case, Left: real part of eigenfunctions, Right: imaginary part of the eigenfunctions. The dominant eigenfunction has zero value imaginary part.

In case the LQR solution is not sufficient, the method can be easily converted into Model Predictive Control (MPC). For the examples in the Simulation Results section, we used MPC for optimal control due to its flexibility to adjust to problem constrains. Given a time horizon of L, we can solve for $V \in \mathbb{R}^{m \times L}$ by minimizing the following cost function, as illustrated in Equation 3.8.

$$J(V) = \sum_{t=0}^{L-1} (z_t^\top C^\top Q C z_t + v_t^\top R v_t) + z_L^\top C^\top Q C z_L \\ subject\ to \begin{cases} v_t \leq u_{max} - u_0 \\ v_t \geq u_m in - u_0 \end{cases} \tag{3.8}$$

3.3 Algorithm Summary

To summarize, a schematic diagram of the DKRC framework is shown in Figure 3.4:

The algorithm of Deep Koopman Representation for Control (DKRC) is listed below, where $L1(\theta)$ ensures a high dimension linear system, $L2(\theta)$ handles the controllability of the system:

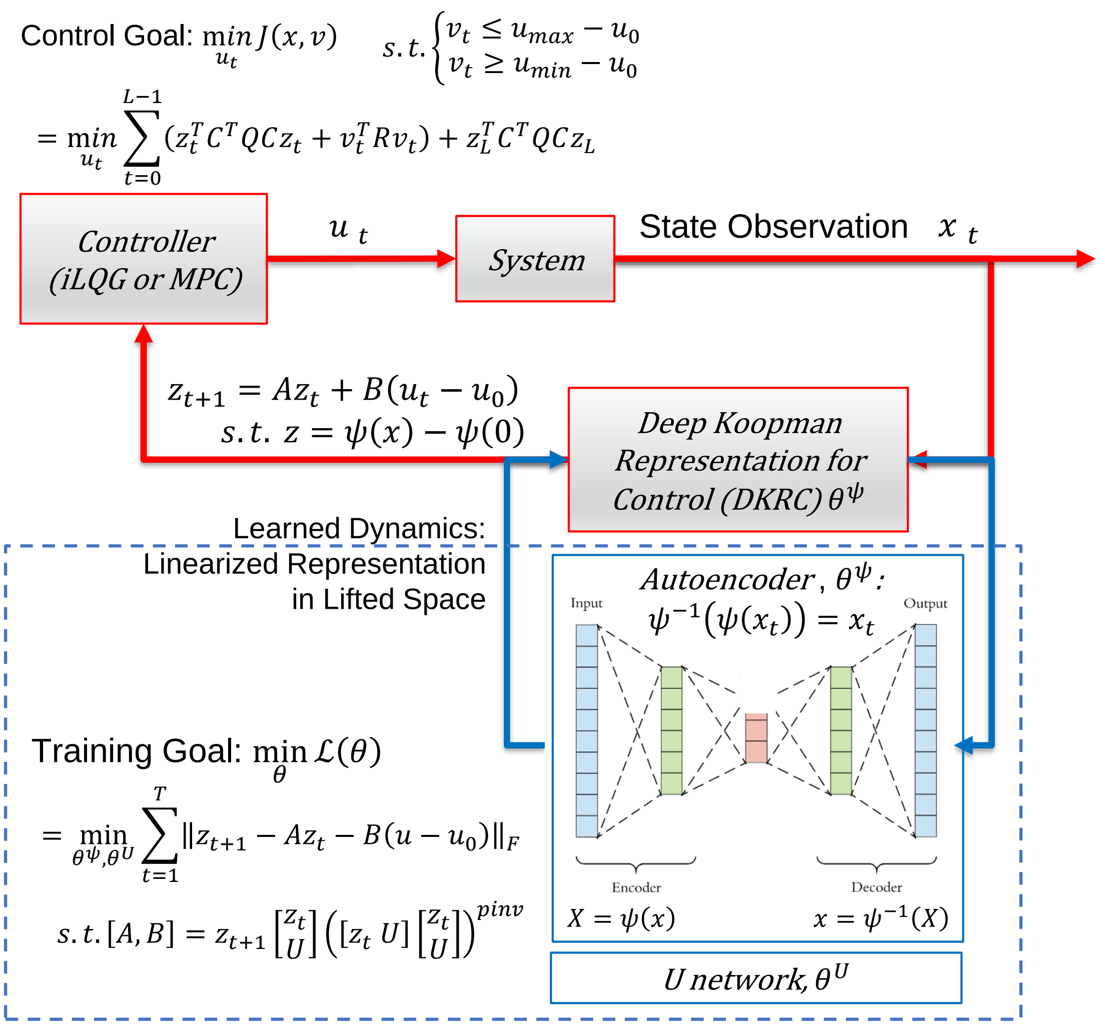

Control Goal: $\min_{u_t} J(x,v)$ $s.t. \begin{cases} v_t \le u_{max} - u_0 \\ v_t \ge u_{min} - u_0 \end{cases}$
$= \min_{u_t} \sum_{t=0}^{L-1} (z_t^T C^T Q C z_t + v_t^T R v_t) + z_L^T C^T Q C z_L$
Controller (iLQG or MPC)
u_t
System
State Observation x_t
$z_{t+1} = Az_t + B(u_t - u_0)$
$s.t.\ z = \psi(x) - \psi(0)$
Deep Koopman Representation for Control (DKRC) θ^ψ
Learned Dynamics: Linearized Representation in Lifted Space
Autoencoder, θ^ψ: $\psi^{-1}(\psi(x_t)) = x_t$
Input
Output
Encoder
Decoder
$X = \psi(x)$
$x = \psi^{-1}(X)$
Training Goal: $\min_\theta \mathcal{L}(\theta)$
$= \min_{\theta^\psi, \theta^U} \sum_{t=1}^{T} \|z_{t+1} - Az_t - B(u - u_0)\|_F$
$s.t. [A, B] = z_{t+1} \begin{bmatrix} z_t \\ U \end{bmatrix} \left([z_t\ U] \begin{bmatrix} z_t \\ U \end{bmatrix} \right)^{pinv}$
U network, θ^U

Figure 3.4: Schematics of DKRC framework

Algorithm 1 Deep Koopman Representation for Control (DKRC)

```
Input: observations: x, control: u
Output: Planned trajectory and optimal control inputs: ($z_{plan}$, $v_{plan}$)
  • Initialization
      1. Set goal position $x^*$
      2. Build Neural Network: $\psi_N(x_t;\theta)$
      3. Set $z(x_t;\theta) = \psi_N(x_t;\theta) - \psi_N(x^*;\theta)$
  • Steps
      1. Set $K = z(x_{t+1};\theta) * z(x_t;\theta)^\dagger$
      2. Set the first loss function $L1$
         $L1(\theta) = \frac{1}{L-1}\sum_{t=0}^{L-1} \parallel z(x_{t+1};\theta) - K * z(x_t;\theta) \parallel$
      3. Set the second loss function $L2$
         $[A, B] = \mathbf{z}_{t+1}\begin{bmatrix}\mathbf{z}_t \\ U\end{bmatrix}\left(\begin{bmatrix}\mathbf{z}_t & U\end{bmatrix}\begin{bmatrix}\mathbf{z}_t \\ U\end{bmatrix}\right)^\dagger$
         $L2(\theta) = (N - rank(controllability(A, B))) + ||A||_1 + ||B||_1$
      4. Train the neural network, updating the complete loss function
         $L(\theta) = L1(\theta) + L2(\theta)$
      5. After converging, We can get system identity matrices A, B, C
         $C = \boldsymbol{X}_t * \psi_N(\boldsymbol{X}_t)^\dagger$
      6. Apply LQR or MPC control with constraints
```

Chapter 4

DKRC Implementation Results

In this chapter, we demonstrate the proposed continuous control method based on deep learning and Koopman operator (DKRC) on several example systems offered in OpenAI Gym [3]. The environments considered here are Inverted Pendulum, Cartpole with Double Pendulum, and Lunar Lander (Continuous Control). In this work, we deploy Model Predictive Control (MPC) on Inverted Pendulum, Cartpole with double pendulum, and adopt LQR for Lunar Lander.

The OpenAI Gym utilizes a Box2D physics engine, which provides a dynamic system for the testing. Gravity, object interaction, and physical contact behavior can be specified in the world configuration file. The dynamics of the simulation environment are unknown to the model. The model needs to learn the dynamics through data collected during training. Each simulation game is randomly initialized to generate initial disturbance.

4.1 Inverted Pendulum

The inverted pendulum environment is a classic 2-dimensional state space environment with one control input, as shown in Equation 4.1.

4.1.1 DKRC solution

A visualization of the simulated environment is also shown in Figure 4.1. The game starts with the pendulum pointing down with a randomized initial disturbance. The goal is to invert the pendulum to its upright position. In order to finish the game, the model need to apply control input to drive the pendulum from start position ($\theta \in (-\pi, 0) \cup (0, \pi)$, $\dot{\theta} = 0$) to goal position ($\theta = 0$, $\dot{\theta} = 0$).

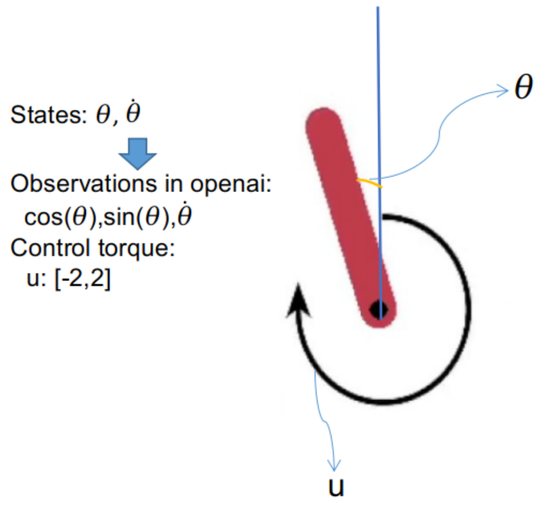

States: θ, $\dot{\theta}$
Observations in openai:
cos(θ),sin(θ),$\dot{\theta}$
Control torque:
u: [-2,2]
θ
u

Figure 4.1: Inverted Pendulum

$$\begin{gathered}\boldsymbol{\chi} = [\cos\theta, \sin\theta, \dot{\theta}] \quad \textit{where } \theta \in [-\pi, \pi], \dot{\theta} \in [-8, 8] \\ \boldsymbol{U} = [u], \quad \textit{where } u \in [-2, 2]\end{gathered} \tag{4.1}$$

In this case, the recorded θ orientation angle is in the form of a pair of cosine and sine values to ensure a smooth transition at $-\pi$ and π. The dynamic governing equation is easy to derive as follows:

$$ml^2\ddot{\theta} = -\gamma\dot{\theta} + mglsin(\theta) + u \tag{4.2}$$

This governing equation is unknown to our model, and the model needs to recover the physics-based on limited time-series data only. Moreover, the angular velocity ($\dot{\theta}$) and the torque input (u) at the shaft are also limited to a small range to add difficulties to finish the game. In most of the testing cases, we found that the pendulum needs to learn a swing back-and-forth motion before being able to collect enough momentum from swinging up to the final vertical upright position. Although optimal control is possible, the trained model needs to adapt to the limitation imposed by the testing environment and make decisions solely based on state observations.

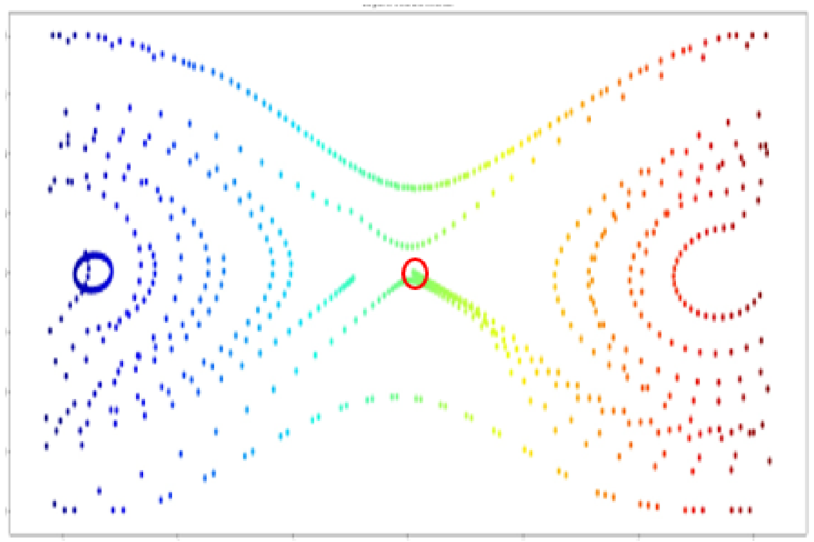


Figure 4.2: Trajectories of Swinging Pendulum under Control using trained DDPG reinforcement learning model. The figure exhibits the trajectories of the tip of the pendulum breaking through Hamiltonian energy level lines to arrive at the goal position ($\theta = 0$, $\dot{\theta} = 0$)

We train the DDPG reinforcement learning model first for benchmark purposes.

Figure 4.2 shows the trajectory of the pendulum in 2D (θ, $\dot{\theta}$) space based on trained DDPG algorithm. The red circle in the center of Figure 4.2 is the goal position, and the concentric lines on both sides correspond to Hamiltonian total energy levels. It is clear that due to the implicit model learned during the reinforcement learning process, the RL method cannot finish the game within a short time and therefore left many failed trails.

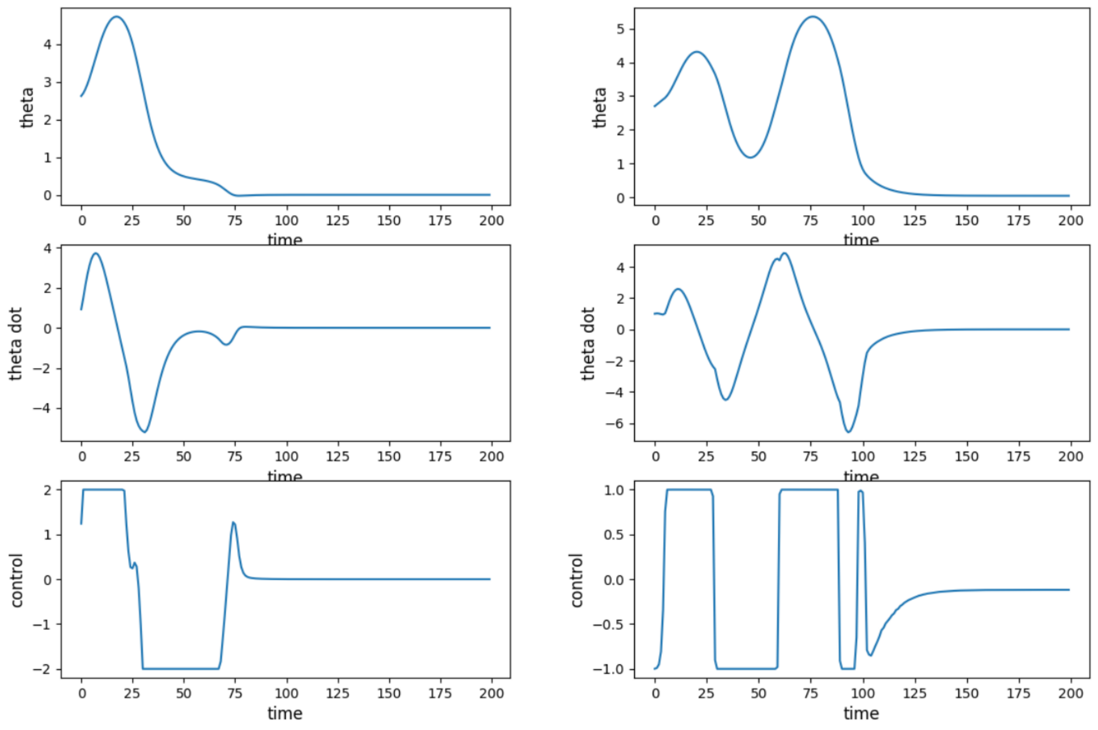


Figure 4.3: Trajectories of Swinging Pendulum under control using DKRC model (left) vs. DDPG RL model (right) initialized at the same state

For comparison between the Reinforcement Learning (DDPG) and DKRC models, we show a single example game in Figure 4.3. The two games were initialized in the same way, and the DKRC model on the left of the figure was able to finish the game much cleaner and faster.

Corresponding to the game shown in Figure 4.3, We are also presenting the state vs. time plot for the game played by DKRC model, as shown in Figure 4.4.

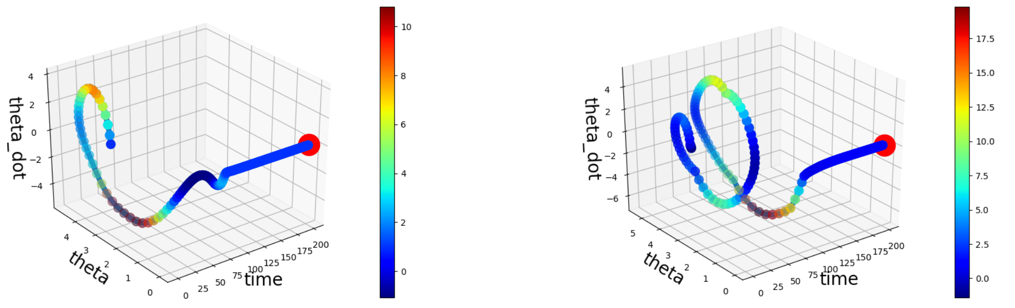


Figure 4.4: Example 3D deployment trajectory from DKRC (left) and Reinforcement Learning (right), colormapped by the magnitude of energy

4.1.2 Stability Test

To measure the stability and the performance of the DKRC solution, we run the Swing-up pendulum game for 100 times and record the reward of each game. As defined in the official OpenAi package, the reward for each step is

$$Reward = -(\theta^2 + 0.1 * \dot{\theta}^2 + 0.001 * u_t^2)$$

where $reward \in [-16.2736044, 0]$, and each game has 200 time steps which means $reward\ each\ game \in [-3224, 0]$.The result is shown in Figure 4.5. It can be inferred from the figure that the total reward of each game is different due to different initial positions as expected. Besides, the DKRC has a 98% successful percentage for 100 games.

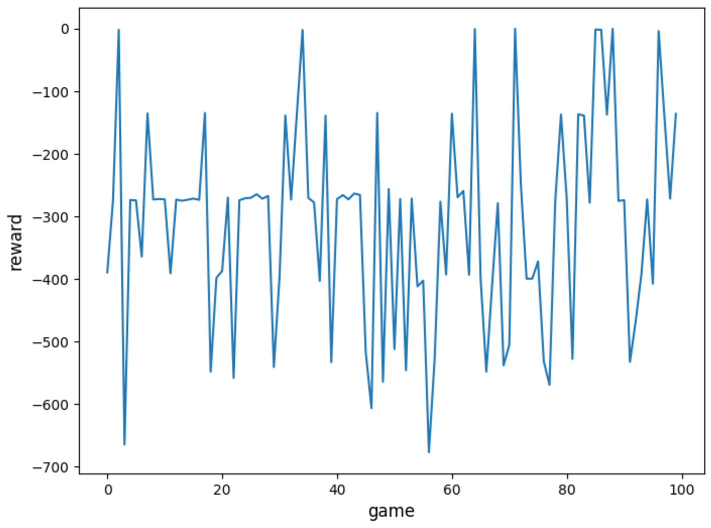


Figure 4.5: 100 games recorded reward using DKRC

4.2 Acrobot

Acrobot is another classic control scenario. We swing a double pendulum with only one actuation at the joint between the two links, which simulates the swing up motion of a human acrobat. Initially, both links point downwards (initial sweeping angles are zero to the world coordinate system). The two links are allowed to swing in any direction at the two joints, without any collision if they are with the same sweep angle. The goal is to swing the end-effector at a height at least the length of one link above the base. The same simulation has been performed mainly for reinforcement learning algorithm developments [36]. A schematic of the environment setup is shown in Figure 4.6

The state consists of the two rotational joint angles and their velocities: [θ_1, θ_2, $\dot{\theta}_1$, $\dot{\theta}_2$]. For the same consideration in the inverted pendulum case, to ensure numerical smoothness during transition from $-\pi$ and π, we use the following state-space observations for training.

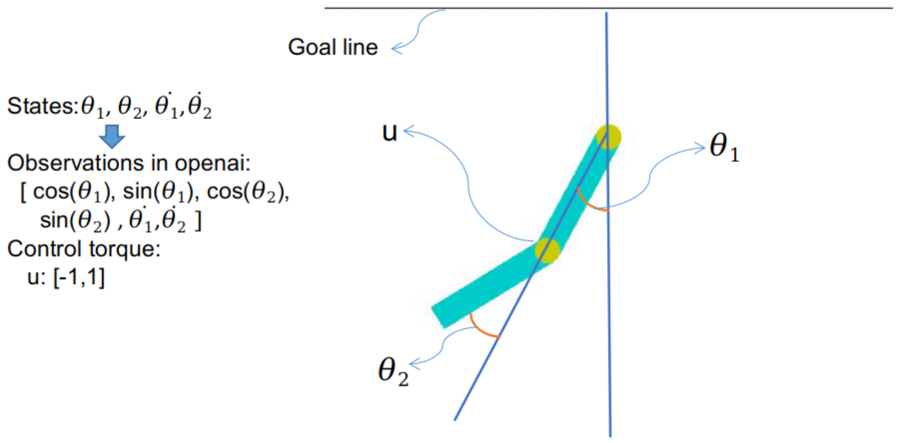

Goal line
States:θ_1, θ_2, $\dot{\theta}_1$,$\dot{\theta}_2$
Observations in openai:
[cos(θ_1), sin(θ_1), cos(θ_2), sin(θ_2) , $\dot{\theta}_1$,$\dot{\theta}_2$]
Control torque:
u: [-1,1]
u
θ_1
θ_2

Figure 4.6: Acrobot

$$\ddot{\theta}_1 = -d_1^{-1}(d_2\ddot{\theta}_2 + \phi_1)$$

$$\ddot{\theta}_2 = (m_2 l_{c_2}^2 + I_2 - \frac{d_2^2}{d_1})^{-1}(\tau + \frac{d_2}{d_1}\phi_1 - m_2 l_1 l_{c_2}\dot{\theta}_1^2 \sin\theta_2 - \phi_2)$$

$$d_1 = m_1 l_{c_1}^2 + m_2(l_1^2 + l_{c_2}^2 + 2 l_1 l_{c_2}\dot{\theta}_1^2 \sin\theta_2 - \phi_2) + I_1 + I_2$$

$$d_2 = m_2(l_1^2 + l_{c_2}^2 + 2 l_1 l_{c_2}\dot{\theta}_1^2 \sin\theta_2 - \phi_2) + I_2$$

$$\phi_1 = -m_2 l_1 l_{c2}\dot{\theta}_2^{\ 2} \sin\theta_2 - 2 m_2 l_1 l_{c2}\dot{\theta}_2\dot{\theta}_1 \sin\theta_2$$

$$+(m_1 l_{c1} + m_2 l_1) g \cos\left(\theta_1 - \frac{\pi}{2}\right) + \phi_2$$

$$\phi_2 = m_2 l_{c2} g \cos\left(\theta_1 + \theta_2 - \frac{\pi}{2}\right)$$

If we want to make $\theta_1 = \pi/2, \theta_2 = -\pi/2, \dot{\theta}_1 = \dot{\theta}_2 = 0$ as equilibrium point, this will require making $\ddot{\theta}_1 = \ddot{\theta}_2 = 0$ and hence we have

$$\phi_2^\star = 0$$

$$\phi_1^\star = (m_1 l_{c_1} + m_2 l_1) g$$

4.2.1 DKRC solution

Here, to take advantage of our MPC controller optimized for continuous control, we customized the environment to use continuous torque at the joint, instead of the OpenAI default discrete input [-1, 0, 1] for the torque. The trajectory of the two links are shown in Figure 4.7. We can finish the game every time according to the criteria listed on the OpenAI leaderboard, as shown in Figure 4.7.

4.2.2 Stability Test

Again, to measure the stability and the performance of the DKRC solution, we run the Acrobot game for 100 times and record the total reward of each game. As defined in the official OpenAi package, the reward for each step is

$$reward = -1,\ if\ not\ terminal\ else\ 0$$

which means $rewardeachgame \in [-400, 0]$.The result is shown in Figure 4.8. It can be inferred from the figure that the total reward of each game is different due to different initial positions as expected. Besides, the DKRC has a 97% successful percentage for 100 games in this scenario.

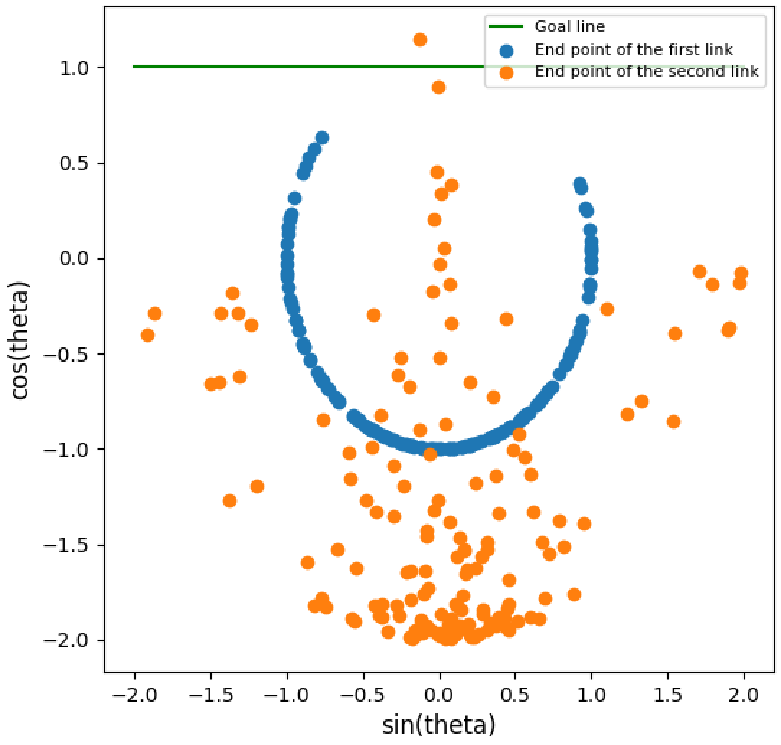


Figure 4.7: DKRC deployment trajectories of Acrobot under Control. As shown in the picture, the green line is goal line for second link.The goal of this game is to touch the green line with the second link

4.3 Double Pendulum on a Cart

A schematic of the double pendulum on a cart environment is shown in Figure 4.9. This testing case is carried out in a customized OpenAI environment that combines two existing OpenAI Gym environments "cartpole" and "acrobot" in one simulation. In this case, a double pendulum is attached to a cart by an un-actuated joint. The cart moves along a frictionless track and can be driven by force in the horizontal direction. The two links of the double pendulum can swing independently without any friction or collision. The pendulum starts upright with a randomized initial disturbance. The

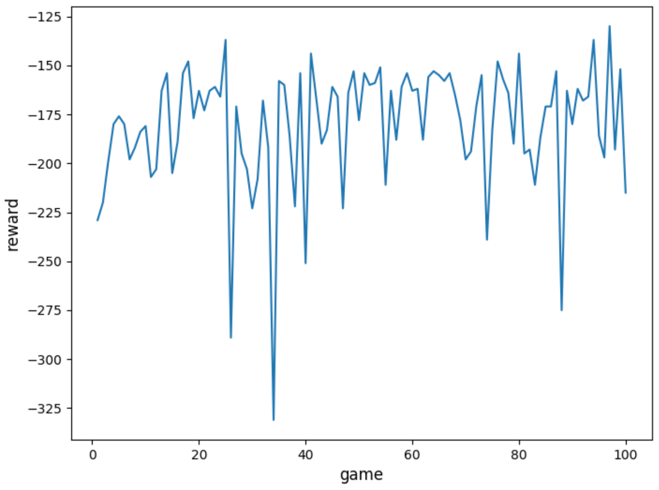


Figure 4.8: 100 games recorded reward using DKRC

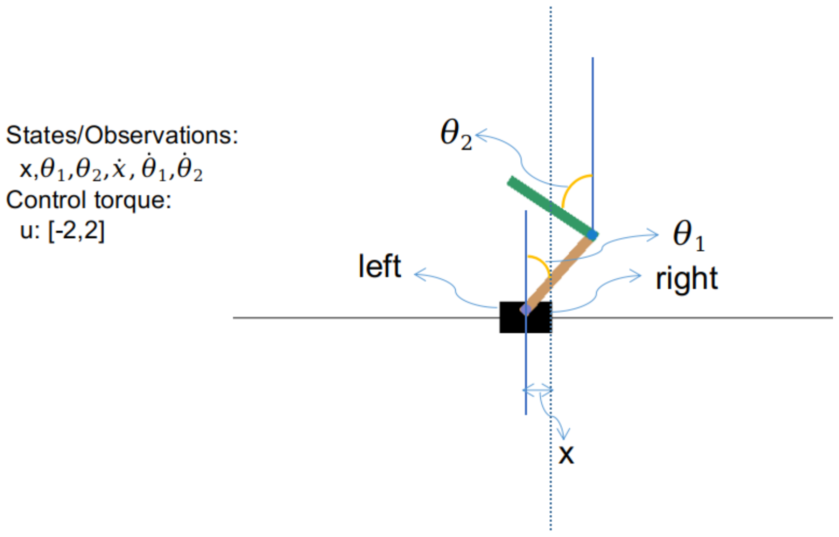


Figure 4.9: Cartpole with double pendulum

goal is to prevent it from falling over by applying force to the cart in the horizontal direction. We use the following state-space observations for training.

$$\boldsymbol{\chi} = [x, \theta_1, \theta_2, \dot{x}, \dot{\theta}_1, \dot{\theta}_2]$$
$$\boldsymbol{U} = [left\ or\ right\ force], \quad where, u \in [-2, 2] \tag{4.3}$$

4.3.1 DKRC solution

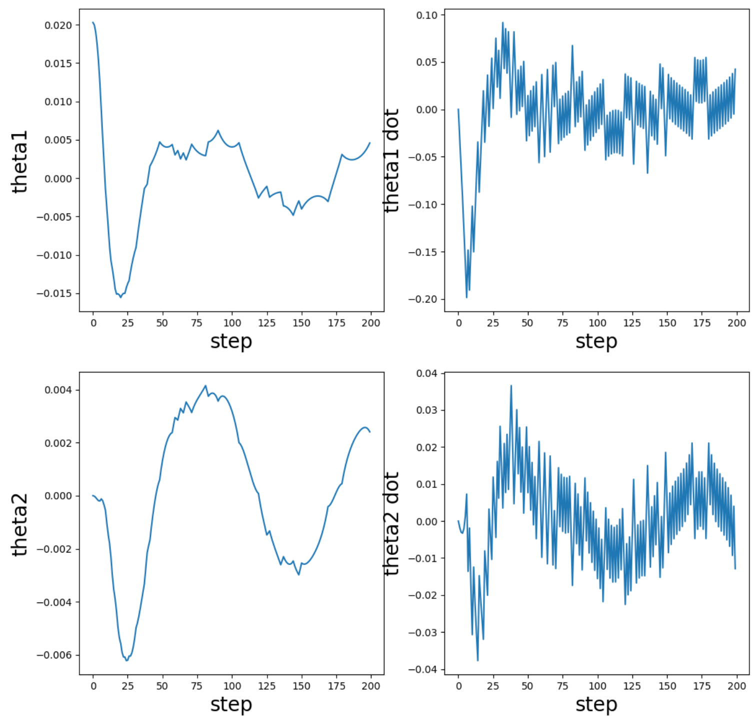


Figure 4.10: Balance keeping process for one game using DKRC

The control input is being applied to the cart and then translated to the motion of the two links. The state measurements of two links are depicted in Figure 4.10. As can be seen in the figure, both the two swing angles (θ_1, θ_2) and corresponding angular velocities ($\dot{\theta}_1$, $\dot{\theta}_2$) are kept within a tight range (axis zoomed-in to show the trend). The state observables oscillate around a value close to 0 throughout the testing period (end of steps), which means that the learned model can maintain a dynamic balance between a cart and an inverted double pendulum.

4.3.2 Stability Test

To measure the stability and performance of the DKRC solution, we run the Swing-up pendulum game for 100 times and record each game's reward. As defined in the official OpenAi package, the reward for each step is

$$Reward = -(\theta_1^2 + \theta_2^2)$$

The result is shown in Figure 4.11. It can be inferred from the figure that the maximum

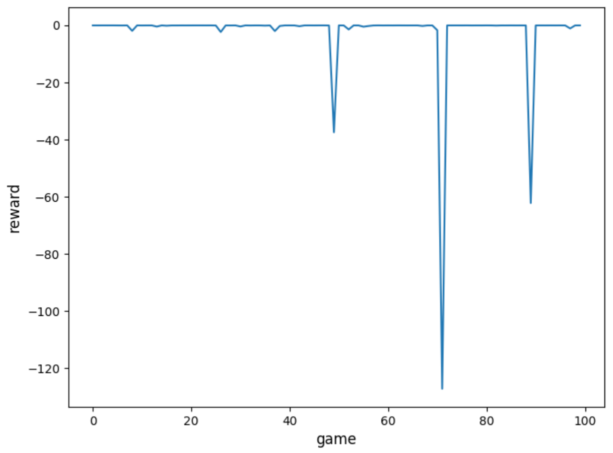


Figure 4.11: 100 games recorded reward using DKRC

reward of this game is 0, and the DKRC has a 97% successful percentage for 100 games.

4.4 Lunar Lander

As depicted in Figure 4.12, the lunar lander environment is a 6-dimensional state space with 2 control inputs. The state observables involve more terms and more closely related to a real-world game environment than previous classic dynamics examples.

$$\begin{aligned} \boldsymbol{\chi} &= [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}] \\ \boldsymbol{U} &= [u_1, u_2], \quad where, u_1 \in [0,1], u_2 \in [-1,1] \end{aligned} \tag{4.4}$$

where x, y are the Cartesian coordinates of the lunar lander, and θ is the orientation angle measured clockwise from the upright position. u_1 represents the main engine on the lunar lander, which only takes positive input. In contrast, the u_2 is the side engine that can take both negative (left engine firing) and positive (right engine firing) values. The goal of this game is to move lunar lander from initial position ($x = 10$, $y = 13$) to the landing zone ($x = 10$, $y = 4$), subject to randomized initial gust disturbance and dynamics specified in the Box2D environment.

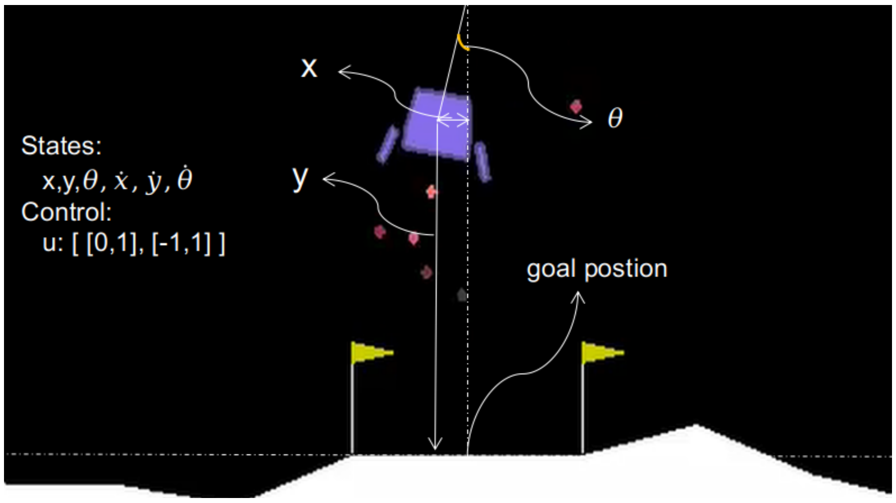


Figure 4.12: Lunar Lander

The exact model for this system cannot be extracted directly from the OpenAI Gym environment but has to be identified either through a data-driven model-based approach (this paper) or model-free approach (e.g., reinforcement learning). A game score is calculated by OpenAI Gym where the system will reward smooth landing within the landing zone (area in between double flags) while penalizing the fuel consumption by engines (U). The proposed method is then to generate the model of system identification. Assuming after the Koopman operator transform, the dynamical system can be considered to be linear. Therefore, model predictive control can be applied. The trajectories of the lunar lander successfully finishing the game are shown in Figure 4.13. Note that the trajectories for the ten games are spread out in the simulation environment and returning back to the same goal position. The reason for the spread-out behavior is that the initialization of each game will randommly assign an initial velocity and the control algorithm need to apply control to offset the drift while keeping balance in the air. In this simulation environment, the DKRC model was demonstrated to be able to learn the dynamics and cope with the unforeseen situation using Model Predictive Control.

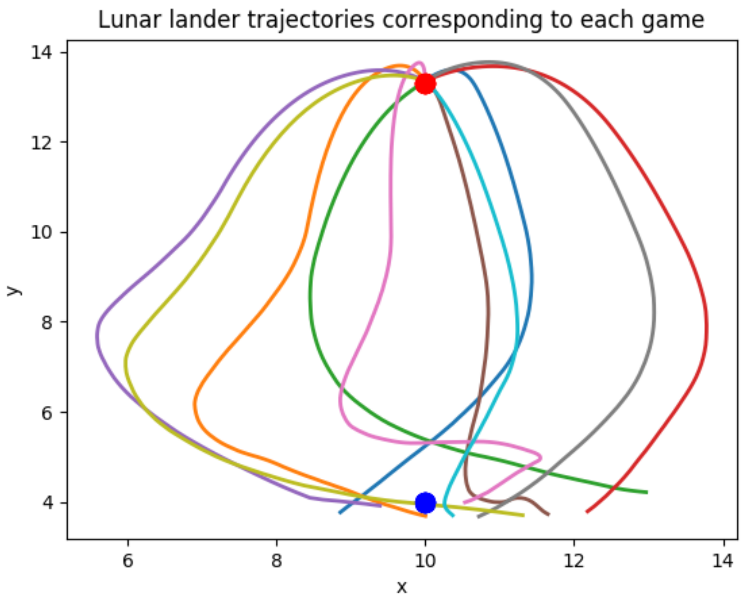


Figure 4.13: DKRC deployment for ten games, Red point: starting point, Blue point: goal position

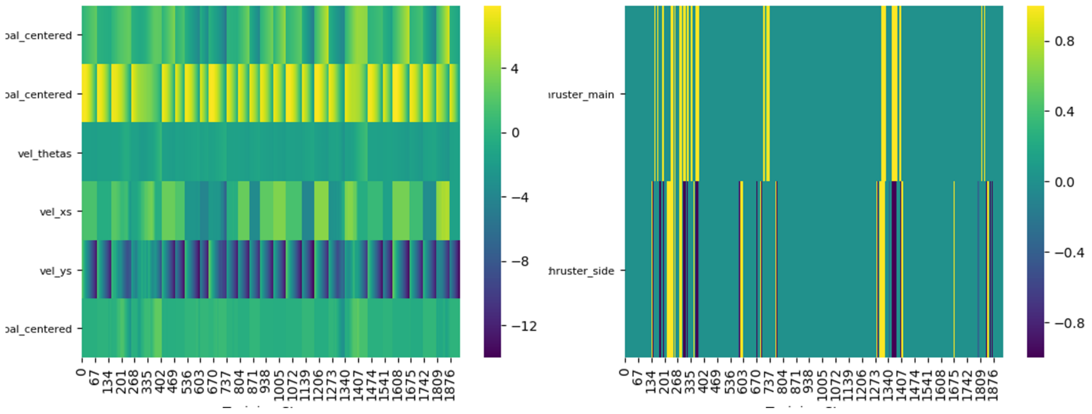


Figure 4.14: Data visualization of the training sample for Lunar Lander environment. The 1876 data pairs are obtained from playing five games. The different color gradient represents different state observations and control inputs collected. Each row on Left figure denotes one of six states; right figure shows the control input from main and side engine thrusts

4.4.1 DKRC solution

The data collection can be obtained by running a reinforcement learning algorithm with random policy together with a random noise generated by the Ornstein–Uhlenbeck process during the data collection procedure. These randomization treatments are implemented to ensure enough exploration vs. exploitation ratio from the Reinforcement Learning model. A sample visualization of the collected data is shown in Figure 4.14 to demonstrate how few amounts of data are needed to train the proposed learning algorithm, which poses a significant advantage compared to state-of-art reinforcement learning methods.

4.5 Experiment Conclusion

In this first part of the thesis, we propose MPC controller designed by Deep Koopman Representation for Control (DKRC) has the benefit of being completely data-driven in learning, whereas it remains to be model-based in control design. The DKRC model is efficient in training and sufficient in the deployment in four OpenAI Gym environments compared to a state-of-the-art Reinforcement Learning algorithm (DDPG).

Chapter 5

Data Driven Control: Model-based vs. Model-free Approach

In this chapter, we compare the proposed DKRC algorithm with another classic model-free control method – Deep Deterministic Policy Gradient (DDPG), in terms of the algorithm control strategies, robustness, and the learned dynamics of DKRC with the classic Euler-Lagrange Linearization method to validate the learning efficiency of DKRC.

5.1 Experiment Setup

To obtain benchmark comparison results, we use the classic 'Pendulum-v0' OpenAI Gym environment [3] to examine the behaviors of controllers built based on DDPG and DKRC for this inverted pendulum problem. The OpenAI Gym is a toolkit designed for developing and comparing reinforcement learning [35]. The inverted pendulum swing-up problem is a classic problem in the control literature, and the goal of the system is to swing the pendulum up and make it stay upright, as depicted in Figure 5.1. Although both the two approaches are data-driven in nature, thus versatile in deployment, We would like to still stick with the classic control problem, which has rich documentation of analytical solution for later comparisons. As shown in the next section, the DKRC approach successfully finishes the control task using a learned dynamics that directly resembles the analytical solution and can also explain the system in lifted dimensions using Hamiltonian Energy Level theorem.

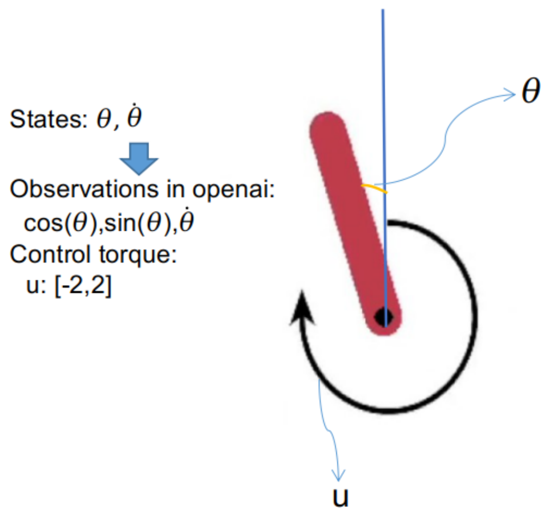

States: $\theta, \dot{\theta}$
Observations in openai:
$\cos(\theta), \sin(\theta), \dot{\theta}$
Control torque:
u: [-2,2]
θ
u

Figure 5.1: Environment visualization

5.1.1 Problem set-up

A visualization of the simulation environment is shown in Figure 4.9. As shown in the picture, the simple system has two states $x = [\theta, \dot{\theta}]$ and one continuous control moment at the joint, which numerically must satisfy $-2 \leq u \leq 2$. This limit in control magnitude is applied with the intention to add difficulty in designing the control strategy. For the default physical parameter setup, the maximum moment that is allowed for the environment will not be sufficient to raise the inverted pendulum to the upright position in one single move. The control strategy needs to learn how to build momentum by switching moment direction at the right time. The OpenAI Gym defines the observations and control in Equation 5.1.

$$\begin{aligned} \boldsymbol{\chi} &= [\cos\theta, \sin\theta, \dot{\theta}], \quad where\ \theta \in [-\pi, \pi], \dot{\theta} \in [-8, 8] \\ \boldsymbol{U} &= [u], \quad where\ u \in [-2, 2] \end{aligned} \tag{5.1}$$

The cost function is designed in Equation 5.2. The cost function tracks the current states (θ) and control input (u). The environment also limits the rotational speed of the up-swing motion, by including a $\dot{\theta}^2$ term in the cost function. By minimizing this cost function, we ask for the minimum control input to ensure the pendulum ends at the upright position with minimum kinetic energy levels. During the comparison, we are using the Equation 5.2 directly in the controller design for the DKRC, whereas we negate cost function and transforms it into a negative reward function, where 0 is the highest reward for the DDPG training. By using this simple simulation environment, we ensure the two approaches are being compared on the same basis with the same reward/cost function definition.

$$reward = \theta^2 + 0.1\dot{\theta}^2 + 0.001u^2 \tag{5.2}$$

The dynamical system of the swing-up pendulum can be analytically solved by Euler–Lagrange method [14], with its governing equation shown in Equation 5.3, where $\theta = 0$ is the upright position, g is the gravitational acceleration, m is the mass of the pendulum, l is the length of the pendulum.

$$\ddot{\theta} = -\frac{3g}{2l}\sin(\theta + \pi) + \frac{3}{ml^2}u \tag{5.3}$$

By converting the second order ordinary differential equation(ODE) in Equation 5.3 to the first order ODE, we can get Equation 5.4.

$$\frac{d}{dt}\begin{bmatrix}\cos\theta \\ \sin\theta \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & \frac{3g}{2l} & 0\end{bmatrix}\begin{bmatrix}\cos\theta \\ \sin\theta \\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}0 \\ 0 \\ \frac{3}{ml^2}\end{bmatrix}u \tag{5.4}$$

We use the default physical parameters defined in the OpenAI Gym, i.e., $m = 1$, $l = 1$, $g = 10.0$, $dt = 0.05$. We discretize the model using zero order hold (ZOH) method [12] with a sampling period dt. As a result, we can achieve the linearized governing Equation 5.5 with observed states y_t. The numerical values for the hyperparameters used in the default problem setting are also listed in Equation 5.6. Also one predictive result is that Euler-Lagrange method only works between a small initial angle assumption as a result of $\sin\theta = \theta$, $\cos\theta = 1$ is validate under small angle assumption.

$$\begin{aligned} x_{t+1} &= A_d x_t + B_d u_t \\ y_t &= C x_t \end{aligned} \tag{5.5}$$

with

$$\begin{gathered} A_d = \begin{bmatrix}0.9988 & -0.04998 & 0 \\ 0.04998 & 0.9988 & 0.05 \\ 0.01875 & 0.7497 & 1\end{bmatrix} \quad B_d = \begin{bmatrix}0 \\ 0 \\ 0.15\end{bmatrix} \\ C = \begin{bmatrix}1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{bmatrix} \end{gathered} \tag{5.6}$$

We use A_d and B_d as the benchmark of learned dynamic models comparison between DKRC and Euler-Lagrange Linearization in later sections.

5.1.2 Parameters of DDPG and DKRC

The following Table (5.1) is the parameters we use to obtain DDPG and DKRC solutions. Both two methods are trained on NVIDIA Tesla V-100 GPUs on an NVIDIA-DGX2 supercomputer cluster.

Table 5.1: Parameters of different methods

Method	Parameter	Value
DDPG	Buffer size	1e6
	Batch size	64
	γ (Discount factor)	0.9
	τ (Target Network Update rate)	0.001
	Target & Actor learning rate	1e-3
	Target & Critic learning rate	1e-2
	Training epochs	5e4
DKRC	Lift dimension	8
	Training epochs	70

The result of the DDPG solution is an Actor neural network with optimal parameters for the nonlinear pendulum system - $\mu(x;\theta^*)$.

The result of the DKRC solution is identity matrices A_{lift}, B_{lift}, C of the lifted space, and a lift neural network $\psi_N(x;\theta)$ for observations of the unknown dynamical system.

5.2 Control Strategies of DDPG and DKRC

We present results from the DKRC vs DDPG by specifying five initialization configuration for the problem. We have choices of defining the starting position and also the initial disturbance in the form of starting angular velocity of the pendulum. In this study, we initialize the pendulum at five different positions: $\theta_0 = \pi$ (lowest position), $\theta_0 = \frac{\pi}{2}$ (left horizontal position), $\theta_0 = -\frac{\pi}{2}$ (right horizontal position), $\theta_0 = \pm\frac{\pi}{18}$ (close to upright position). The initial angular velocity of pendulum at different initial positions is $\dot{\theta}_0 = 1rad/s$ (clockwise). For better visualization, we map the angle from $[-\pi, \pi]$ to $[0, 2\pi]$, where the upright position (goal position) is always achieved at $\theta = 0$ and $\dot{\theta} = 0$. The result is shown in Figure 5.2− 5.5.

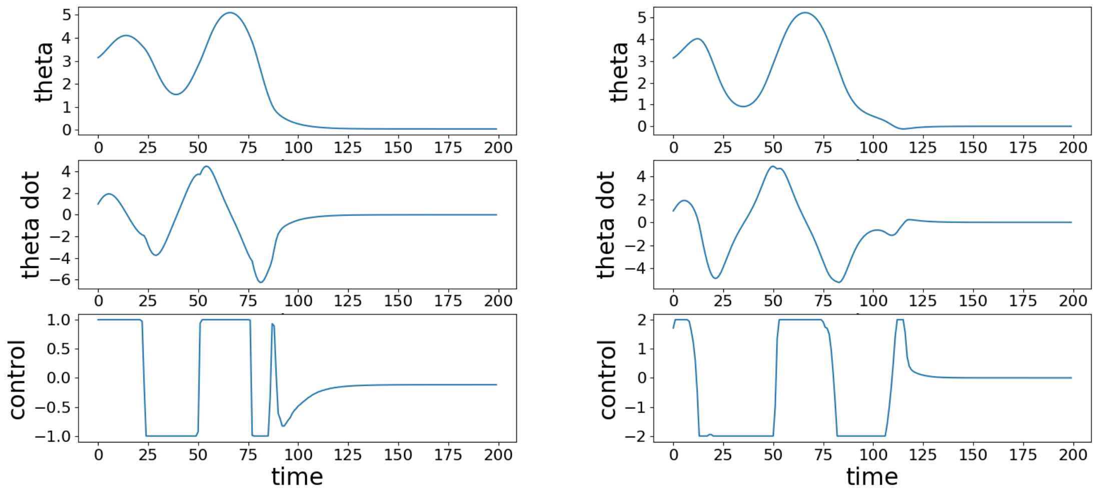


Figure 5.2: Trajectories of Swinging Pendulum implementing DDPG model (left) vs. DKRC model (right) initialized at the lowest position

Figure 5.2− 5.6 show that DDPG and DKRC have similar control strategies in most initial positions. DDPG needs less time to arrive at the goal position than DKRC when the pendulum is initialized on the right side. It tends to use smaller control torque as a result of that DDPG has constraints term for control torque. Still, DDPG never succeeds in getting an absolute upright position, i.e., at the final position, a non-zero control input is always required to sustain a small displacement away from the goal position. On the contrary, DKRC can achieve a precise goal position with

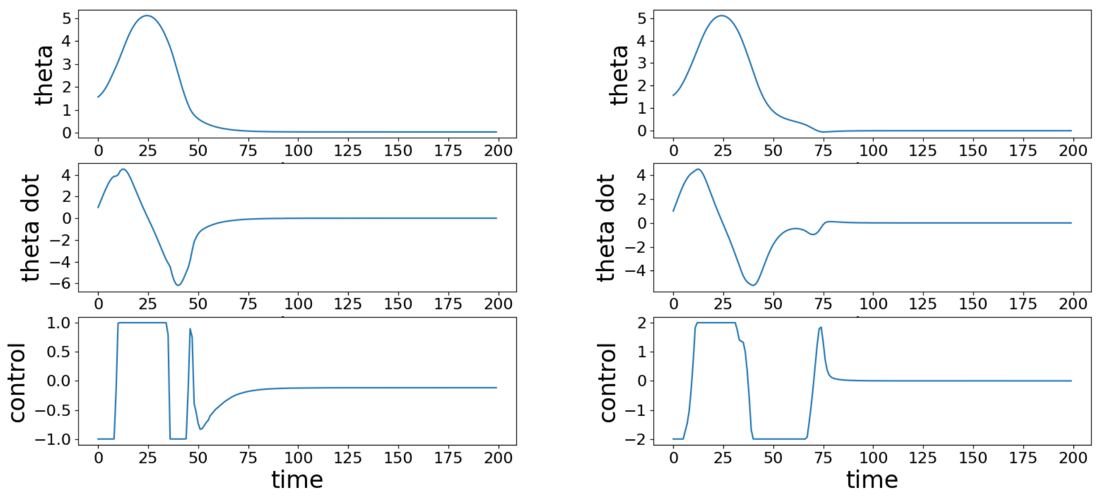


Figure 5.3: Trajectories of Swinging Pendulum implementing DDPG model (left) vs. DKRC model (right) initialized at the left horizontal position

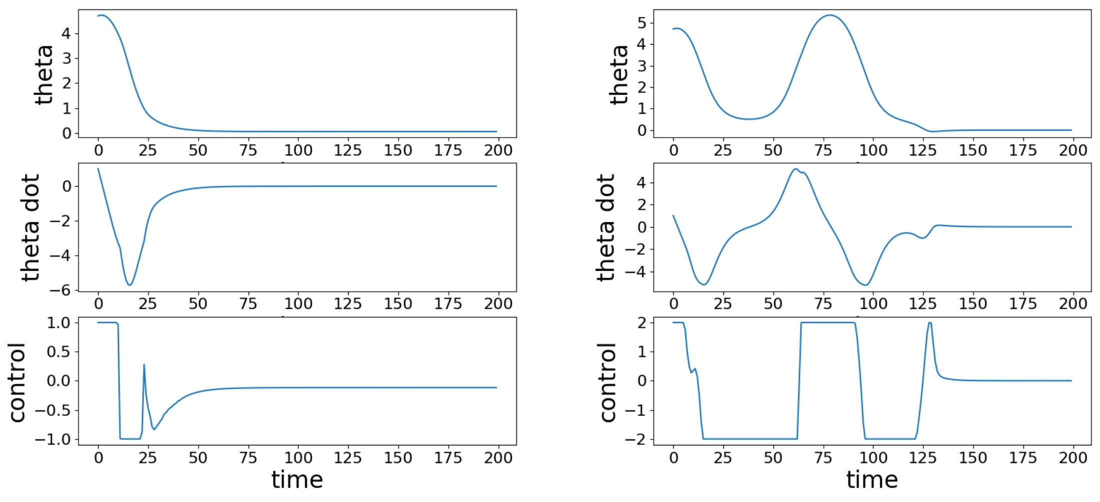


Figure 5.4: Trajectories of Swinging Pendulum implementing DDPG model (left) vs. DKRC model (right) initialized at the right horizontal position

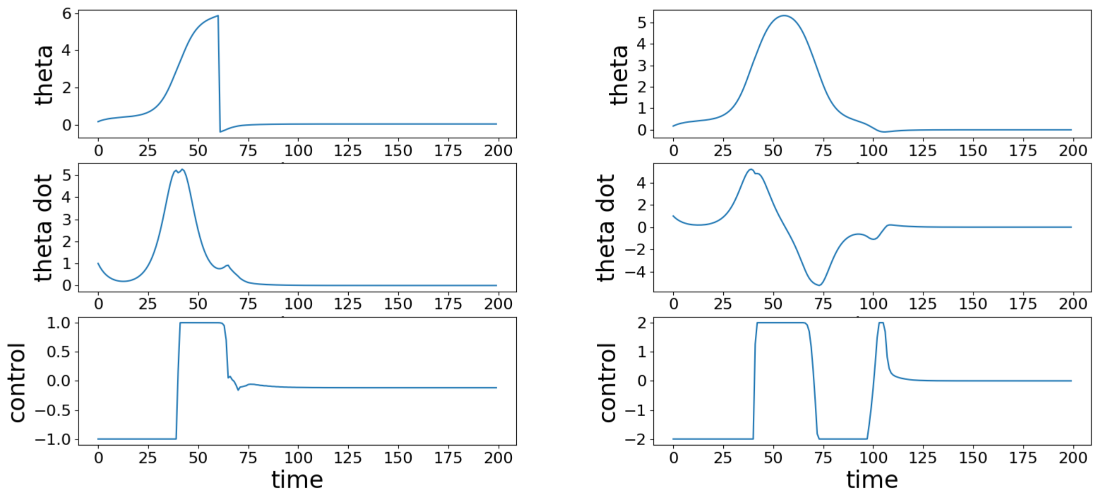


Figure 5.5: Trajectories of Swinging Pendulum implementing DDPG model (left) vs. DKRC model (right) initialized at the close left upright position

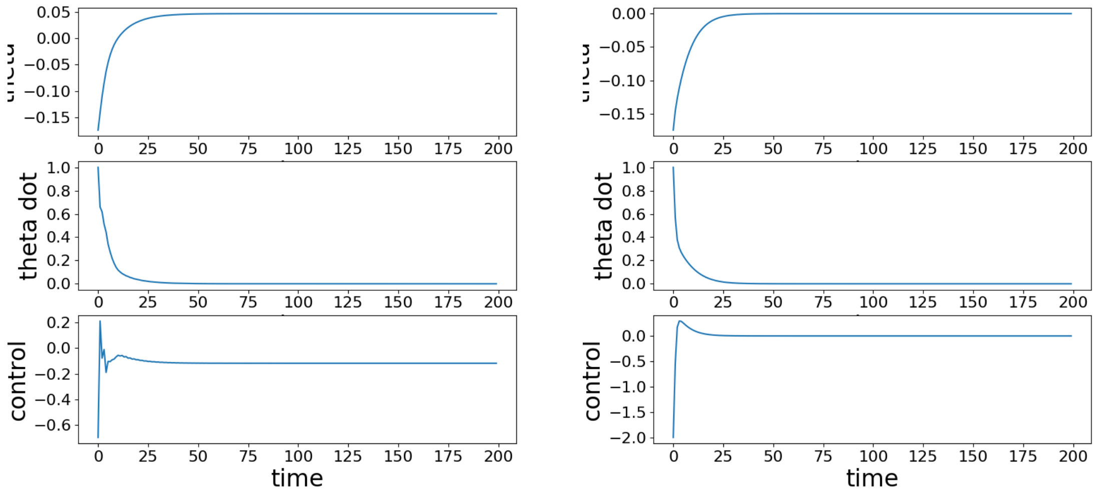


Figure 5.6: Trajectories of Swinging Pendulum implementing DDPG model (left) vs. DKRC model (right) initialized at the close right upright position

much less training time than DDPG. Once a proper dynamical system model can be learned directly from data, it makes more sense to execute control using model-based controller design such as MPC.

Another way to show the differences in control strategies is by plotting the measured trajectories during repeated tests. In Figure 5.7, we test the pendulum game for 50 games with a total of 10000 time-steps utilizing both methods (DDPG & DKRC) solutions. In this comparison, we plot the measurements of $\dot{\theta}$ vs. θ on a 2D

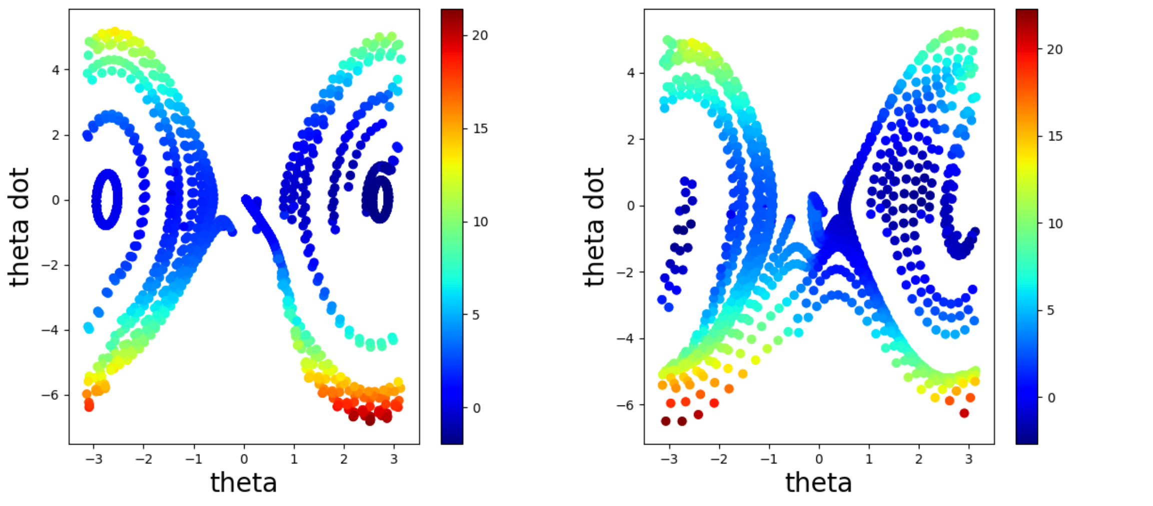


Figure 5.7: 50 pendulum games data recorded using DDPG model (left) vs. DKRC model (right), color mapped by energy

basis, colored by the magnitude of the cost function defined in Equation 5.2. The goal is to arrive at the goal position ($\dot{\theta} = 0$ and $\theta = 0$, the center of each plot) as quickly as possible. Figure 5.7 shows that DDPG tends to drive the states into "pre-designed" patterns, and execute similar control strategies for the 50 games. Therefore the data points on the left subplot appear to be less than the one on the right. DKRC, on the other hand, tends to exhibit different control behavior due to the local replanning using MPC. The result of the local replanning is that it generates multiple trajectories solving the 50 games with different random initializations. During this comparison, we illustrate that DDPG is indeed deterministic, which is a good indicator of the system's reliability. However, we want to point out that the robustness of the system will also benefit from a local replanner available under different initializations or disturbances since the "pre-designed" patterns are learned from past experience, therefore, cannot guarantee a viable solution for unforeseeable situations when we move onto more complex systems.

By transforming the observed states, we can also show the relationship between the designed control and the energy levels in the system. Consequently, our control goal is to achieve the lowest energy level of the system in terms of lowest magnitudes in both kinetic energy and potential energy. In Figure 5.8, we present the Hamiltonian energy level plots with respect to the measured states (θ, $cos(\theta)$, $sin(\theta)$, and $\dot{\theta}$) for the forced frictionless pendulum system, each colored by the same energy level scale. The energy term used in forced pendulum is defined as $\frac{1}{2}\dot{\theta}^2 + cos(\theta) + u$. Figure 5.8 shows

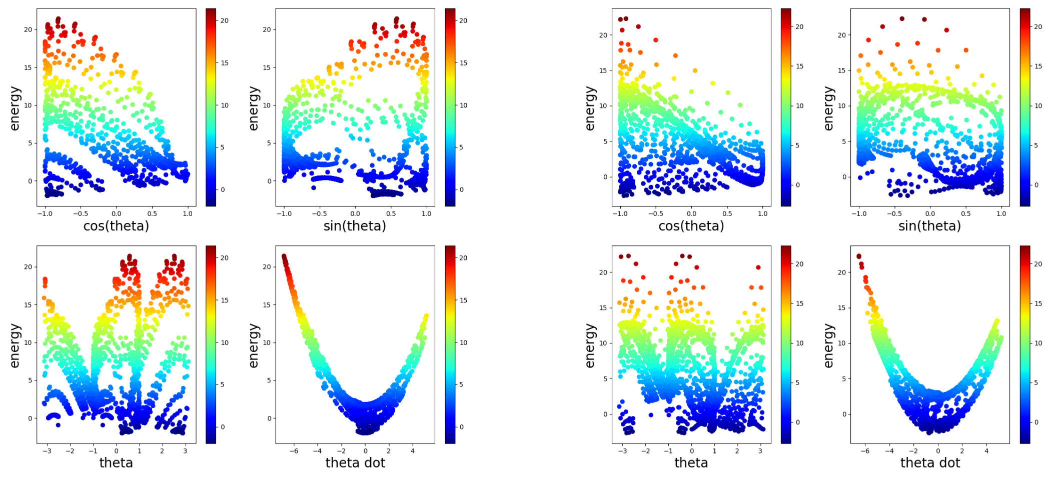


Figure 5.8: DDPG(left) vs. DKRC(right): Recorded Observations & Energy

that DKRC's trajectories are concentrated in lower energy areas compared to DDPG, which means it intends to minimize the energy directly. This behavior also resembles many design strategies used in classic energy-based controller design approaches.

5.3 Control Visualization using Decoder Neural Network

Another benefit of having a model-based controller design is the interpretability of the system. We can preview the design trajectory since our MPC controller implements the receding horizon control. By deploying one pendulum game, we obtain one planned trajectory for the linear system in the lifted dimension space. To get the comparison between the planned trajectory and the measured trajectory in the state space, we utilize the decoder neural network obtained during the training to map the trajectory in higher-dimensional space (8-dimension for inverted pendulum) to the lower-dimensional space (2-dimension). The comparison between such recovered trajectory planning and the actual trajectory is presented in Figure 5.10. In this figure, we again mapped the θ in the range of 0 to 2π for visualization purposes, whereas the goal position is still at the $\dot{\theta} = 0$ and $\theta = 0$ position.

In this work we build an auto-decoder neural network to map the planned high-dimensional trajectory back to the non-lifted states space. The auto-decoder is trained after we get a DKRC solution, and the relationship between the auto-decoder and DKRC is shown in Figure 5.9.

In this example case, we initialize the pendulum close to the goal position but give it a moderate initial angular velocity pointing in the opposite direction to the goal position. The DKRC can plan a simple trajectory with continuous control. During the process, we executed MPC multiple times and used feedback measurements

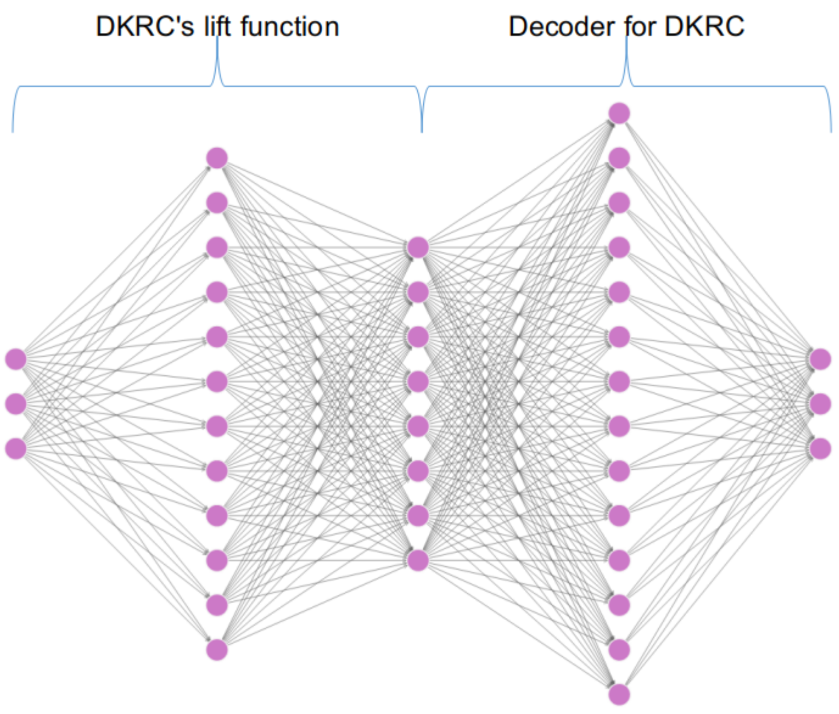


Figure 5.9: Schematics of autoencoder neural network sturcture of DKRC

to improve the design trajectory. The planned trajectory is being closely followed except certain locations close to 0 and π position. The outlier behavior comes from the neural network treatment for the discontinuity and does not pollute the efficient control for the entire problem. It is worth noticing that, to arrive at this result, we learn the unknown nonlinear dynamics using a purely data-driven approach, and we have to go through an encoding-decoding process to recover the planned trajectory. It is promising to state that the use of Koopman representation for nonlinear control can help with system interpretability, which is currently an active research area.

To verify whether DKRC is valid in a more complex environment than Inverted pendulum, we also deploy it in the Lunar Lander - continuous control environment of OpenAI Gym. A simple explanation of 'Lunar Lander' is exhibited in Figure 5.11. The control goal is to guide the lunar lander to arrive at the landing zone as smoothly

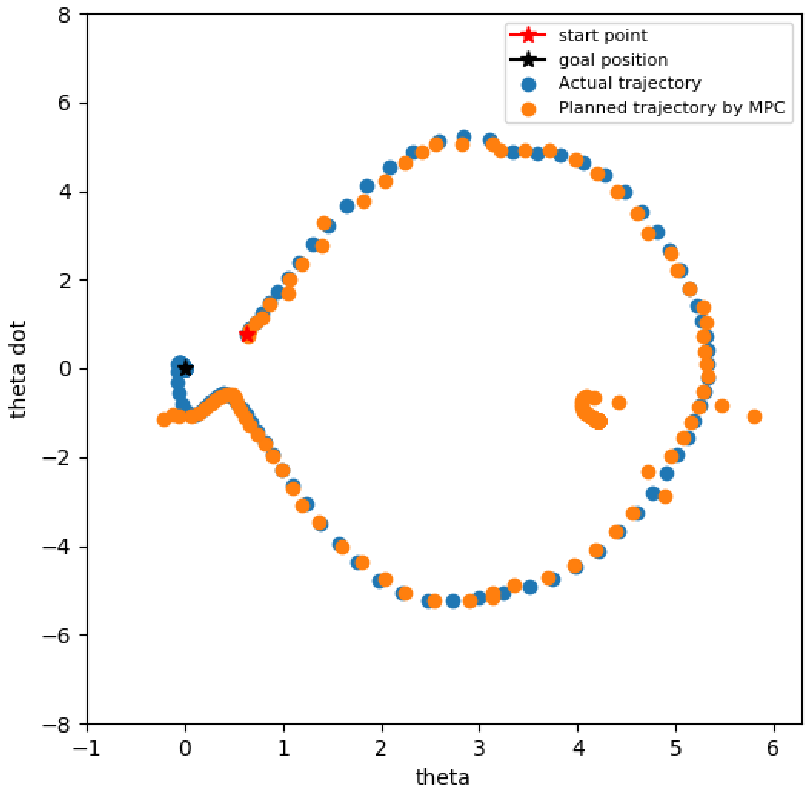


Figure 5.10: DKRC's planned trajectory and actually executed trajectory in Inverted pendulum

as possible [13]. The system is also unknown and must be learned from data. We implement the DKRC framework and use MPC for trajectory planning, and the result is shown in Figure 5.12.

The actual trajectory measured in state-space is shown in hollow red circles. The planned trajectory is colored by the distance away from the originally planned location. It is evident that in the region where the originally planned location is in the immediate vicinity (dark blue color), the actual trajectory is following the plan very precisely. We implement a finite-horizon control during each MPC planning phase. We plan and execute control with several steps beyond the current state as

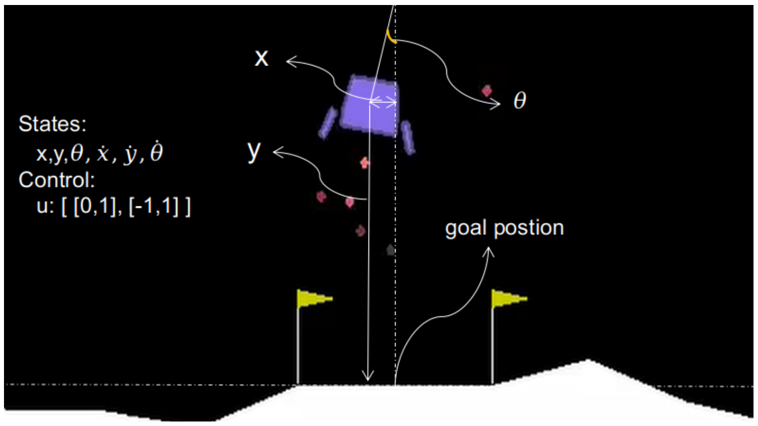


Figure 5.11: Lunar Lander Environment

displayed in lighter green color. The actual trajectory slowly deviates from the planned trajectory when it starts to drift away from the originally planned location. This behavior is expected since we are relying on open-loop control during those finite horizon plannings. The actual trajectory and the projected trajectory merge again once the next round of the MPC control is executed.

In this figure, we demonstrate that the deviation from the planned trajectory and the measured trajectory is from the open-loop planning, rather than from the error introduced while passing the state inputs through the encoder-decoding neural network. The proposed structure is capable of recovering the designed strategy in higher-dimensional space and improving the system's interpretability.

5.4 Robustness Comparison

To compare the robustness of these two methods, we introduce noises to the state measurements and observe the control outcome from DDPG and DKRC. The noise is designed as multiplying the states with a noise ratio, which is randomly selected from range $[0.6, 1]$ during deployment. In this test, we assume the learned dynamics

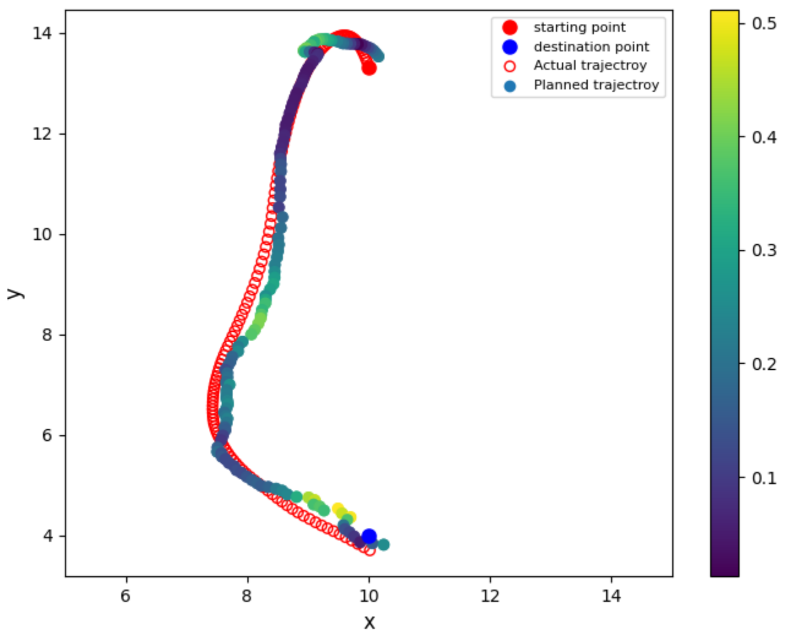


Figure 5.12: DKRC's planned trajectory and actual executed trajectory in Lunar Lander, color mapped by the distance between the planned point and the executed point

from DDPG and DKRC are not affected by the noise; only the observations during deployment are affected, representing a cyber-security attack during the operation phase. The new state inputs would be $x = x * noise$ in this scenario. The result for five repeat games is shown in Figure 5.13. Each line with the same color in the subplot represents measurements from a single game. Even with a high noise ratio, DKRC can succeed in the control task for three out of five games, whereas DDPG fails every game in Figure 5.13. In this test, we demonstrate that the DKRC is capable of designing a control strategy based on noisy data and continuously adjust the control based on the feedback control loop so that it is still robust in a noisy environment. The robustness of DKRC is another advantage compared to a neural-network-based control system, which relies heavily on state measurement accuracy.

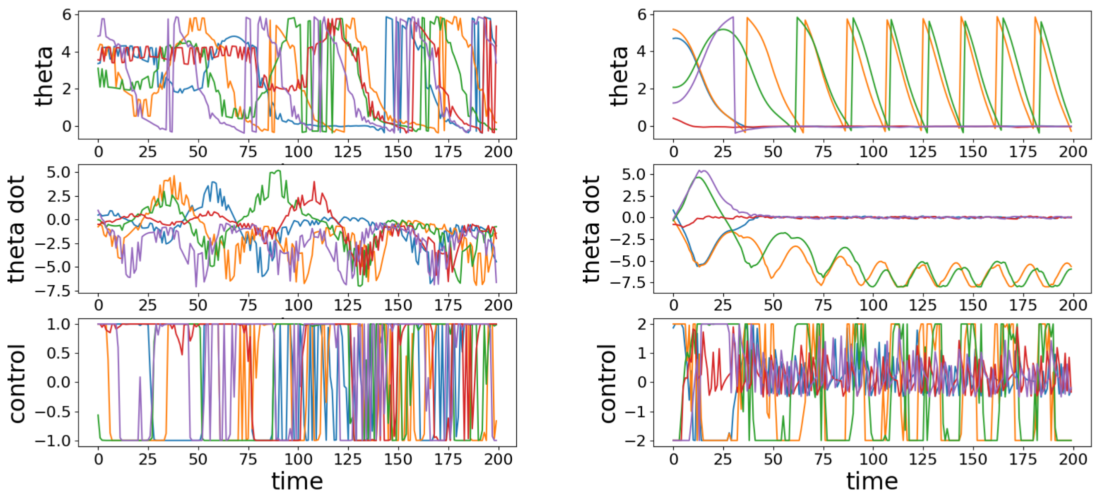


Figure 5.13: Five pendulum games with input states noise using DDPG (left) vs. DKRC (right) solutions

5.5 Learned dynamics of DKRC compared to Euler-Lagrange analytical solution

To illustrate the validity of the learned dynamics using DKRC, we present benchmark comparisons between the proposed DKRC framework and the Euler-Lagrange method's analytical solutions. As previously shown in Equation 5.5 and 5.6, we have obtained the identity matrices for the linearized system using ZOH method. We are making the same assumption for the linearized system in the lifted-dimensional space by the DKRC method. For comparison with different dimension embeddings, we pick the matrices corresponding to the top n eigenvalues from the A and B obtained through DKRC. For the inverted pendulum problem, we collect K time-step data (only need $K = 2000$ data points) to obtain Koopman representation of the system. The resultant dimension is $N = 8$ ($K \gg N$). The learned dynamics A, B of the lifted linear system $\psi_N(x|\theta)$ are shown in the following matrices

$$A_{DRKC} = \left[\begin{array}{ccc|ccccc} 1.01 & -0.06 & -0.01 & -0.08 & -0.08 & 0.06 & -0.01 & -0.01 \\ 0.00 & 0.98 & -0.04 & 0.02 & 0.06 & -0.07 & 0.01 & 0.02 \\ 0.06 & 0.05 & 0.93 & 0.06 & 0.00 & 0.00 & -0.03 & 0.02 \\ \hline 0.03 & -0.00 & -0.01 & 0.97 & -0.04 & 0.05 & -0.01 & -0.01 \\ 0.11 & 0.02 & -0.03 & 0.01 & 0.98 & 0.05 & -0.07 & 0.01 \\ 0.04 & 0.05 & 0.01 & 0.02 & 0.01 & 1.01 & -0.05 & 0.04 \\ 0.01 & 0.014 & 0.02 & 0.01 & 0.08 & -0.04 & 0.96 & 0.04 \\ -0.00 & -0.01 & -0.02 & -0.00 & 0.08 & -0.08 & -0.00 & 1.00 \end{array}\right] \quad B_{DRKC} = \begin{bmatrix} 0.00014 \\ 0.00018 \\ -0.00024 \\ 0.00017 \\ 0.00021 \\ 0.00038 \\ 0.00008 \\ -0.0001 \end{bmatrix}$$

To measure the similarity of A_d (in Equation 5.6), A_{DDPG}, A_{DKRC} we achieved, we adopt the Pearson correlation coefficient(PCC) method [11] in Equation 5.7, where matrices with bar operator, e.g. $\bar{A}$, is the sample mean of that matrix. The result $r(A, B) \in [0, 1]$ represents the correlation between the two matrices. Two matrices (A and B) are more similar when r is closer to 1.

$$r(A,B) = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}} \tag{5.7}$$

To compare the similarity of these two matrices with different dimensions, we extract the left top 3×3 part of the A_{DKRC}. We achieve a correlation score of $r(A_d, A_{DKRC}) = 0.8926$, which means the lifted linear system of DKRC is very similar to the analytical system model solved by Euler-Lagrange method. This high correlation score indicates that the data-driven Koopman representation of the system can reveal the intrinsic dynamics purely based on data samples.

On a separate note, we do not expect the two results are exactly the same since by lifting the system to a higher-dimensional space, we have more neurons in DKRC neural networks to store the system information that was not included in the previous comparison. In addition, we also deploy the linearization model obtained by the Euler-Lagrange method with MPC to the same OpenAI Gym environment to examine the effectiveness by direct linearization without lifting. Unlike DKRC or DDPG, which can work for any arbitrary initial configuration, control designed by the Euler-Lagrange method only works when the pendulum's initial position is between $[-23.4°, 23.4°]$ with small initial angular velocity disturbance. A sample comparison between Euler-Lagrange MPC and DKRC is exhibited in Figure 5.14.

As shown in Figure 5.14, DKRC only spends around 20% time of Euler-Lagrange linearization method to make pendulum converge to the upright position.

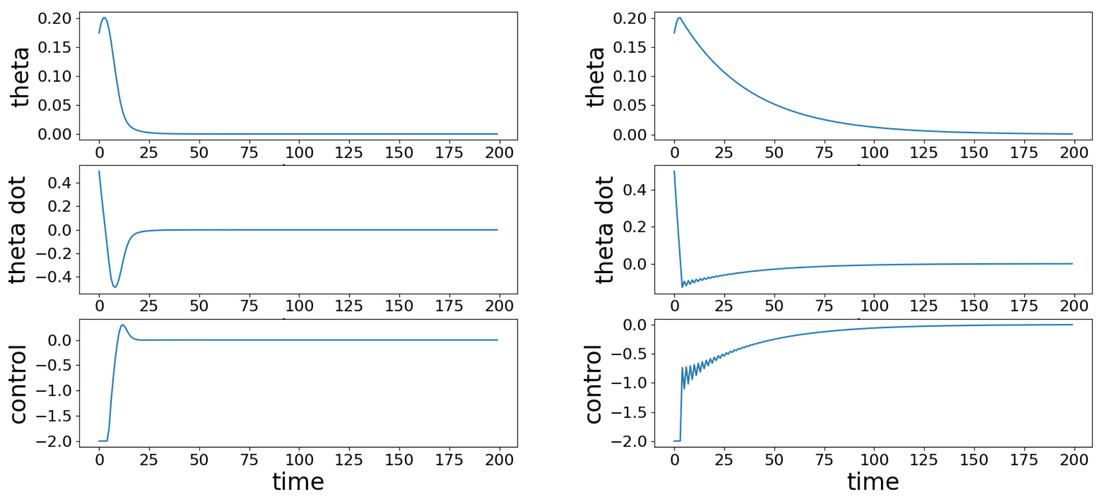


Figure 5.14: One pendulum game using DKRC model (left) vs. Euler-Lagrange method (right); Both tests initialized at $\theta_0 = \frac{\pi}{18}$, $\dot{\theta}_0 = 0.5$

5.6 Experiment Conclusion

This second part of the thesis provides a systematic discussion of two different data-driven control frameworks: Deep Deterministic Policy Gradient (DDPG) and Deep Learning of Koopman Representation for Control (DKRC). Both the two methods are data-driven and adopt neural networks as central architecture, but the controller design is model-free for DDPG, whereas DKRC utilizes a model-based approach. Our experiments in a simple swing-up pendulum environment demonstrate the different solutions achieved by DKRC and DDPG. The DKRC method can achieve the same control goal as effective as the DDPG method but requires much less training epochs and training samples (70 epochs vs. 5×10^4 epochs). Due to the physics model-based nature, DKRC provides better model interpretability and robustness, which are both critical for real-world engineering applications.

Chapter 6

Conclusions

- This study proposes a data-driven control method with actively learned dynamics from neural networks. With a lifting approach adopted from Koopman Operator theory, we can achieve a linearized system representation in high-dimensional space without known mathematical governing equations. The proposed algorithm is suitable to design optimal control for complex dynamical systems in real-world engineering problems.

- When compared to the model-free method like the Reinforcement Learning (RL) method, the proposed method exhibits many advantages in training efficiency (only takes 70 training epochs compared to typically dozens of thousands of epochs for RL algorithms. In real-world time, it typically takes days to train a good RL algorithm for a relatively complicated problem). Compared to RL, it is more robust under disturbances generated by noise injection, benefiting from a model-free online control method with more control strategies.

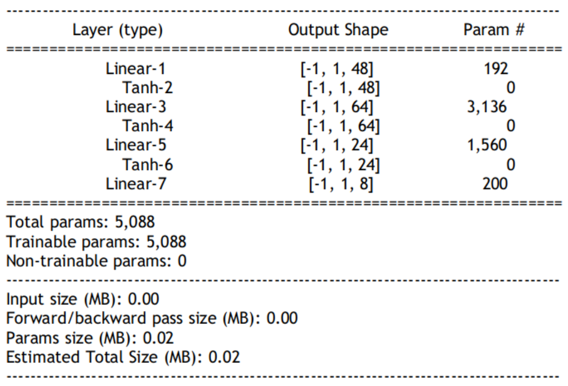
- During the training of the Koopman representation of the system, an educated guess is needed to determine the lifting dimension. To choose a proper lifting dimension, we add an additional loss function in the cost function definition to capture the size of lifting dimensions that can make the second loss function keep full rank in the controllability test.

Appendices

Appendix A Neural network structures of DKRC solutions for different dynamic systems

A.1 Neural Network Structure for Inverted Pendulum

Figure 1 shows the $\psi_N(x;\theta)$ structure used to solve single swing-up pendulum problem, where $x \in \mathbb{R}^3$, $N = 8$.

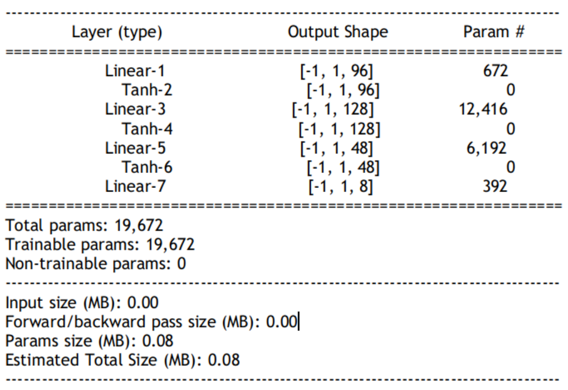


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1              [-1, 1, 48]             192
              Tanh-2              [-1, 1, 48]               0
            Linear-3              [-1, 1, 64]           3,136
              Tanh-4              [-1, 1, 64]               0
            Linear-5              [-1, 1, 24]           1,560
              Tanh-6              [-1, 1, 24]               0
            Linear-7               [-1, 1, 8]             200
================================================================
Total params: 5,088
Trainable params: 5,088
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.02
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
```

Figure 1: Neural Network Structure For Inverted Pendulum

A.2 Neural Network Structure for Balancing Double Pendulum on a Cart

Figure 2 shows the $\psi_N(x;\theta)$ structure used to balance a double pendulum on a cart, where $x \in \mathbb{R}^6$, $N = 8$.

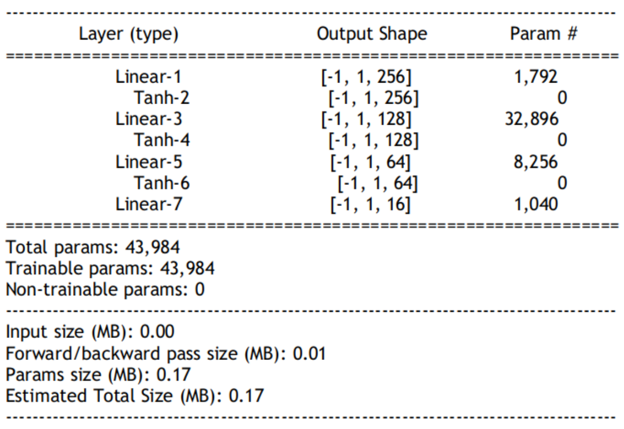


```
--------------------------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 96]             672
              Tanh-2               [-1, 1, 96]               0
            Linear-3              [-1, 1, 128]          12,416
              Tanh-4              [-1, 1, 128]               0
            Linear-5               [-1, 1, 48]           6,192
              Tanh-6               [-1, 1, 48]               0
            Linear-7                [-1, 1, 8]             392
================================================================
Total params: 19,672
Trainable params: 19,672
Non-trainable params: 0
--------------------------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.08
Estimated Total Size (MB): 0.08
--------------------------------------------------------------------------------
```

Figure 2: Neural Network Structure For Double Pendulum Balancing on a Cart

A.3 Neural Network Structure for Lunar Lander

Figure 3 shows the $\psi_N(x;\theta)$ structure used to control the lunar lander to reach the goal position, where $x \in \mathbb{R}^6$, $N = 16$.

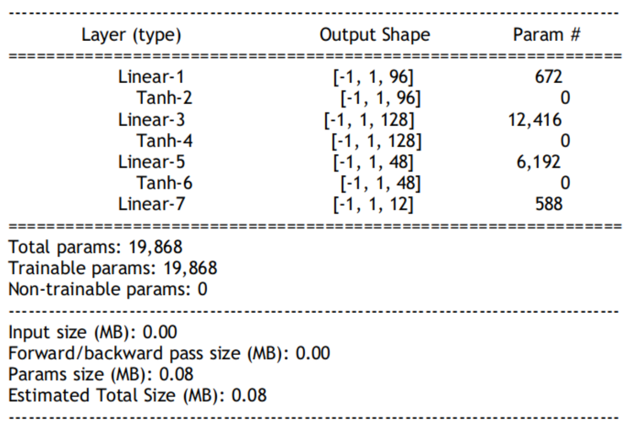


```
--------------------------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1              [-1, 1, 256]           1,792
              Tanh-2              [-1, 1, 256]               0
            Linear-3              [-1, 1, 128]          32,896
              Tanh-4              [-1, 1, 128]               0
            Linear-5               [-1, 1, 64]           8,256
              Tanh-6               [-1, 1, 64]               0
            Linear-7               [-1, 1, 16]           1,040
================================================================
Total params: 43,984
Trainable params: 43,984
Non-trainable params: 0
--------------------------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.17
Estimated Total Size (MB): 0.17
--------------------------------------------------------------------------------
```

Figure 3: Neural Network Structure For Lunar Lander Control

A.4 Neural Network Structure for Acrobot

Figure 4 shows the $\psi_N(x;\theta)$ structure used to swing up the acrobot to the defined position, where $x \in \mathbb{R}^6$, $N = 12$.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 96]             672
              Tanh-2                [-1, 1, 96]               0
            Linear-3               [-1, 1, 128]          12,416
              Tanh-4               [-1, 1, 128]               0
            Linear-5                [-1, 1, 48]           6,192
              Tanh-6                [-1, 1, 48]               0
            Linear-7                [-1, 1, 12]             588
================================================================
Total params: 19,868
Trainable params: 19,868
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.08
Estimated Total Size (MB): 0.08
----------------------------------------------------------------
```

Figure 4: Neural Network Structure For Acrobot Swing Up

Appendix B Technical Parameters of Controllers

B.1 Swing-up Pendulum

Pendulum controller

Controller: Model predictive control (MPC)

Time horizon: 6

Time execution: 1

Goal position: Upright ($\theta = 0$, $\dot{\theta} = 0$)

B.2 Acrobot

Acrobot controller

Controller: Model predictive control (MPC)

Time horizon: 8

Time execution: 1

Goal position: touching the defined line ($height = length\ of\ pendulum$)

B.3 Double Pendulum on a Cart

Double Pendulum on a Cart Controller

Controller: Model predictive control (MPC)

Time horizon: 24

Time execution: 1

Goal position: Upright position ($\theta_1 = 0, \theta_2 = 0, \dot{\theta_1} = 0, \dot{\theta_2} = 0$)

B.4 Lunar Lander

Lunar Lander Controller

Controller: Finite Linear–quadratic regulator (LQR)

LQR planning horizon: 138

Goal position: Middle point between two flags ($x = 10, y = 4, \theta = 0, \dot{\theta} = 0$)

Bibliography

[1] Olivier Sigaud Arnaud de Froissard de Broissia. Actor-critic versus direct policy search: a comparison based on sample complexity. *arXiv:1606.09152*, 2016.

[2] Alexander Broad, Todd Murphey, and Brenna Argall. Learning models for shared control of human-machine systems with unknown dynamics. *arXiv preprint arXiv:1808.08268*, 2018.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] Marko Budisic, Ryan Mohr, and Igor Mezic. Applied koopmanism. *Chaos*, 22:047510–32, 2012.

[5] IOFFE S. & SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[6] F. Capitanescu D. Ernst, M. Glavic and L. Wehenkel. Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):517–529, 2012.

[7] TUYLS K. & BABUŠKA R DE BRUIN T., KOBER J. The importance of experience replay database composition in deep reinforcement learning. *Deep RL workshop at NIPS*, 2015.

[8] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016. To appear.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[10] A. Eskandarian. Handbook of intelligent vehicles. *Springer London*, 2012.

[11] David Freedman, Robert Pisani, and Roger Purves. Statistics. *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[12] Phil Goddard. Zero order hold with variable time step. *MATLAB Central File Exchange*, 2020.

[13] Yiqiang Han, Wenjian Hao, and Umesh Vaidya. Deep learning of koopman representation for control. *submitted to IEEE CDC2020, forthcoming*, 2020.

[14] Michiel Hazewinkel. Lagrange equations (in mechanics)", encyclopedia of mathematics. *Springer Science+Business Media B.V. / Kluwer Academic*, 1994.

[15] Bowen Huang, Xu Ma, and Umesh Vaidya. Feedback stabilization using koopman operator. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 6434–6439. IEEE, 2018.

[16] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of koopman eigenfunctions for control. *arXiv preprint arXiv:1707.01146*, 2017.

[17] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven approximations of dynamical systems operators for control. In *The Koopman Operator in Systems and Control*, pages 197–234. Springer, 2020.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the national academy of sciences of the united states of america*, 17(5):315, 1931.

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

[21] 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.

[22] H.J. Ferreau M. Diehl and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. *Nonlinear model predictive control*, pages 391–417, 2009.

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

[24] Alexandre Mauroy and Igor Mezić. Global stability analysis using the eigenfunctions of the koopman operator. *IEEE Transactions on Automatic Control*, 61(11):3356–3369, 2016.

[25] Alexandre Mauroy, Yoshihiko Susuki, and Igor Mezić. *Introduction to the Koopman Operator in Dynamical Systems and Control Theory*, pages 3–33. Springer International Publishing, Cham, 2020.

[26] I. Mezić. Spectral properties of dynamical systems, model reductions and decompositions. *Nonlinear Dynamics*, 2005.

[27] Kavukcuoglu Koray Silver David Graves Alex Antonoglou Ioannis Wierstra Daan Mnih, Volodymyr and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv:1312.5602*, 2013.

[28] Sebastian Peitz and Stefan Klus. Koopman operator-based model reduction for switched-system control of pdes. *arXiv preprint arXiv:1710.06759*, 2017.

[29] 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.

[30] Arvind Raghunathan and Umesh Vaidya. Optimal stabilization using lyapunov measures. *IEEE Transactions on Automatic Control*, 59(5):1316–1321, 2014.

[31] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1, 06 2014.

[32] HEESS N. DEGRIS T. WIERSTRA D. & RIEDMILLER M. SILVER D., LEVER G. Deterministic policy gradient algorithms. *Proceedings of the 30th International Conference in Machine Learning*, 2014.

[33] Amit Surana and Andrzej Banaszuk. Linear observer synthesis for nonlinear systems using koopman operator framework. In *Proceedings of IFAC Symposium on Nonlinear Control Systems*, Monterey, California, 2016.

[34] Yoshihiko Susuki and Igor Mezic. Nonlinear koopman modes and coherency identification of coupled swing dynamics. *IEEE Transactions on Power Systems*, 26(4):1894–1904, 2011.

[35] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *MIT press*, 2018.

[36] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[37] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[38] Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra Timothy P. Lillicrap, Jonathan J. Hunt. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[39] U. Vaidya and P. G. Mehta. Lyapunov measure for almost everywhere stability. *IEEE Transactions on Automatic Control*, 53:307–323, 2008.

[40] U. Vaidya, P.G. Mehta, and U. Shanbhag. Nonlinear stabilization via control Lyapunov measure. *IEEE Transactions on Automatic Control*, 55:1314–1328, 2010.

[41] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, page 279–292, 1992.

[42] Wikipedia contributors. Model predictive control — Wikipedia, the free encyclopedia, 2020. [Online; accessed 1-July-2020].

[43] Matthew O Williams, Maziar S Hemati, Scott TM Dawson, Ioannis G Kevrekidis, and Clarence W Rowley. Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49(18):704–709, 2016.

[44] 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.

[45] 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.

[46] Matthew O Williams, Clarence W Rowley, and Ioannis G Kevrekidis. A kernel-based approach to data-driven koopman spectral analysis. *arXiv preprint arXiv:1411.2260*, 2014.

[47] 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.

[48] Pengcheng You, John Pang, and Enoch Yeung. Deep koopman controller synthesis for cyber-resilient market-based frequency regulation. *IFAC-PapersOnLine*, 51(28):720–725, 2018.

[49] Nasser L. Azad Yuan Lin, John McPhee. Comparison of deep reinforcement learning and model predictive control for adaptive cruise control. *arXiv:1910.12047*, 2019.