University of California
Santa Barbara

The Koopman Operator for Representation and Optimal Control of Entity-Based Systems

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Madeline Tara Blischke

Committee in charge:

Professor João Hespanha, Chair
Professor Bassam Bamieh
Professor Igor Mezić
Professor Andrew Teel

June 2026

The Dissertation of Madeline Tara Blischke is approved.

Professor Bassam Bamieh

Professor Igor Mezić

Professor Andrew Teel

Professor João Hespanha, Committee Chair

June 2026

The Koopman Operator for Representation and Optimal Control of Entity-Based Systems

by

Madeline Tara Blischke

To my friends

Acknowledgements

I would first like to thank my advisor, Professor João Hespanha, for all of his support over these last five years, and for his patience and understanding when things were difficult. I also thank Professor Bassam Bamieh, Professor Igor Mezić, and Professor Andrew Teel for having served on my committee.

Next I would like to thank Raphael Chinchilla, Murat Erdal, Sean Anderson, Zeki Duman, and Jair Certório, all of whom I got to know during our time together in our research group. I would also lie to extend thanks to my other peers in the department and the CCDC, who I got to know through classes, seminars, and other events.

Finally, I'd like to thank all of my other friends that I made during my time at the University. It's hard to imagine having made it all this way without all of you who have and who have been there and have supported me throughout this journey.

Curriculum Vitæ

Madeline Tara Blischke

Education

2026	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
2023	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara.
2021	B.S.E. in Electrical Engineering, University of Michigan, Ann Arbor.

Publications

M. Blischke and J. P. Hespanha, "Optimal Control of Entity-Based Systems via Koopman Representations with Product Density Observables," *IEEE Open Journal of Control Systems*, pp. 1–15, 2026.

M. Blischke and J. P. Hespanha, "Learning Switched Koopman Models for Control of Entity-Based Systems," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, Dec. 2023, pp. 6006–6013.

A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "A general language-based framework for specifying and verifying notions of opacity," *Discrete Event Dynamic Systems*, vol. 32, no. 2, pp. 253–289, Jun. 2022.

A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "A dynamic obfuscation framework for security and utility," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, May 2022, pp. 236–246.

A. Wintenberg, M. Blischke, S. Lafortune, and N. Ozay, "Enforcement of k-step opacity with edit functions," in *2021 60th IEEE Conference on Decision and Control (CDC)*, Dec. 2021, pp. 331–338.

Abstract

The Koopman Operator for Representation and Optimal Control of Entity-Based Systems

by

Madeline Tara Blischke

The Koopman operator, which describes a dynamical system via a linear representation that can be approximately learned from data, allows application of linear control techniques to nonlinear systems. One may consider these aspects of system representation and controller design separately, but successful application will ultimately require a fusion of both. This thesis studies contributions to both aspects.

The predictive accuracy of a Koopman-based representation is heavily dependent on the chosen set of lifting functions, termed observables. Identifying an appropriate lifting is a challenging domain-specific problem. We focus our study on a class of systems that are "entity-based systems" that we introduce and formally define. The states of such systems may be subdivided into a variable number of components, which we refer to as "entities". In the Koopman operator framework, we introduce a type of observable that describes a notion of the density of these entities. We discuss limitations of these types of observables, and further consider how products of these densities allow us to capture a richer class of interactions between entities.

Given a Koopman representation for a dynamical system, we may study a nonlinear optimal control problem via a linear problem on a lifted state space. We show that the optimal cost-to-go to this linear problem has a piecewise linear form with respect to the lifted state. A robust formulation, in which we minimize the worst-case cost with respect to bounded errors in the initial state and Koopman dynamics, retains a piecewise affine

structure. Due to the combinatorial nature of the problem, the number of possible input sequences grows exponentially in the horizon, and so we additionally provide a heuristic pruning algorithm that aims to approximate the cost-to-go by restricting the search space to a much smaller subset.

Contents

Chapter 1

Introduction

It is common to approach dynamical systems via a state-based perspective, in which one studies how the state of the system evolves in time. An alternative, equally valid approach, is via an operator-theoretic perspective, in which one instead studies how functions of the state evolve. Whereas the state-space dynamics may be nonlinear, the evolution of these functions is always governed by the linear *Koopman operator*, named as such after B. O. Koopman who first proposed and studied it in the 1930s [33], [34].

Study of the Koopman operator has seen a resurgence of interest in recent decades, thanks to its applicability in spectral analysis [44], [47], and its connection to modern computational tools such as dynamic mode decomposition [58]. An extensive body of literature has since been developed, as covered by numerous comprehensive surveys [14], [15], [21], [43].

Despite its linearity, the Koopman operator has been shown to overcome limitations one may normally expect from linear systems. For instance, it may be used to extend the classical Hartman-Grobman theorem [23], [26] from a local result to one that is valid on the entire basin of attraction [38]. It has also been shown to be capable of describing systems with isolated equilibria [3], [5], though this does come with strict limitations as

discussed in [41].

Beyond analysis and prediction for autonomous systems, there is great interest in applying the Koopman operator to controlled systems [18], [53], [61]. Two main formal extensions have been studied, via function spaces on infinite input sequences as in [35] and via input-parameterized operators as in [27]. Nonetheless, both have equivalent expressiveness [28], and in particular are able to describe the linear models [13], [56], [57], bilinear models [22], [62], and switched system models [54] widely considered in the literature. These system models are immediately amenable to common control approaches such as linear quadratic regulation [13], [32] and model predictive control [19], [35]. More specialized approaches have also been explored, such as control Lyapunov functions [31], control barrier functions [20], eigenstructure assignment [30], and active learning strategies [1]. Koopman-based methods have seen success in a wide range of applications, including robotics [60] (particularly in soft robotics [11], [24]), power grid stabilization [37], traffic control [40], biological processes [29], chemical processes [51], and vehicular applications [42].

Regardless of what control approach is taken, its performance will be heavily dependent on the accuracy of the Koopman model, which in turn is heavily dependent on the choice of lifting functions. Identifying an appropriate lifting is a challenging domain-dependent problem, especially for systems that do not fit cleanly into a classical state space description. Consider arcade games as an example, which have become a standard benchmark in reinforcement learning due the wide range of complex combinatorial problems they present [6], [49]. These games often consist of numerous objects that may appear and disappear over time, a property also found in other systems of interest such as vehicles in traffic networks or individuals in dynamic populations. Compositional Koopman operators proposed in [39] provide generalizability between systems with different numbers of objects, but do not allow for the number of objects to dynamically vary over

time. Alternative representations considered in the literature such as dimension-varying systems [16], [48] or open multi-agent systems [17] may be adaptable to these circumstances, but the problem of identifying observables remains. This motivates further exploration of flexible representation structures that are amenable to Koopman-based methods.

Once we have a Koopman model that describes the system, complex nonlinear control problems are reduced to linear ones. For systems with discrete inputs, which may be modeled as a switched linear system, their inherent combinatorial nature remains a significant challenge [72]. Many approaches in the literature rely on the positive (semi-)definite structure of a quadratic cost function [2], [68], [69], [71]. In the Koopman setting, however, it is most natural to employ linear approximations of cost, as this any nonlinear function may be represented this way if included as an observable in the model. Even with compatible methods, such as the branch-and-bound approach proposed in [67], or mixed-integer program formulations such as in [8], scalability issues arise due to the high dimensionality of Koopman models. This motivates the study of novel approaches that are tailored to our specific needs.

The remainder of this thesis is structured as follows. Chapter 2 serves as an introductory presentation of Koopman operator. Chapter 3 defines a class of entity-based systems, and discusses how they can be represented in the Koopman framework. Chapter 4 discusses and demonstrates contributions to optimal control using input-parameterized Koopman models. Chapter 5 provides brief concluding remarks and a discussion of potential directions for future research.

Chapter 2

The Koopman Operator with Inputs and Outputs

This chapter reviews background material on the Koopman operator, including its definition, basic properties, and data-driven approximation. We then present extensions to actuated systems as considered in the literature, and describe the specific form of input-output representations we will use in the remainder of this work.

2.1 Autonomous Systems

We first discuss the case of autonomous dynamical systems, i.e. those that are not driven by an external input. Consider a system with state $x \in \mathcal{X}$ that evolves in discrete-time such that the next state x^+ is given by

$$x^+ = f(x) \tag{2.1}$$

for some map $f : \mathcal{X} \to \mathcal{X}$. For now, we may consider the states $x \in \mathcal{X}$ to be abstract, non-numeric elements, as described in [46].

In contrast to the state-based dynamics described by (2.1), the Koopman operator describes the dynamics via functions of the state $\phi : \mathcal{X} \to \mathbb{C}$. Such functions are referred to in the literature as *observables*.

Definition 2.1. *Let $\mathcal{F}$ be a function space of observables $\phi : \mathcal{X} \to \mathbb{C}$. We define the* ***Koopman operator*** *$\mathcal{K} : \mathcal{F} \to \mathcal{F}$ associated with* (2.1) *as*

$$\mathcal{K}\phi := \phi \circ f, \qquad \forall \phi \in \mathcal{F}. \tag{2.2}$$

That is, the Koopman operator maps an observable ϕ to a new observable $\mathcal{K}\phi$ whose value is that of ϕ one time step in the future, since

$$(\mathcal{K}\phi)(x) = \phi(f(x)) = \phi(x^+).$$

Whereas the underlying system (2.1) may be nonlinear, the Koopman operator defined in (2.2) is always linear on the space of observables.

Proposition 2.2. *The Koopman operator $\mathcal{K}$ is a linear operator on $\mathcal{F}$.*

Proof: Let $\phi_1, \phi_2 \in \mathcal{F}$ and $\alpha_1, \alpha_2 \in \mathbb{C}$. Then

$$\mathcal{K}(\alpha_1\phi_1 + \alpha_2\phi_2) = (\alpha_1\phi_1 + \alpha_2\phi_2) \circ f = \alpha_1\phi_1 \circ f + \alpha_2\phi_2 \circ f = \alpha_1\mathcal{K}\phi_1 + \alpha_2\mathcal{K}\phi_2$$

■

2.1.1 Spectral Properties

As a linear operator, one may study the spectrum of the Koopman operator, such as eigenvalues and eigenfunctions.

Definition 2.3. *Let $\varphi \in \mathcal{F}$ be a nonzero observable for which there exists $\lambda \in \mathbb{C}$ such that*

$$\mathcal{K}\varphi = \lambda\varphi.$$

*Then we say φ is an **eigenfunction** of the Koopman operator corresponding to the **eigenvalue** λ.*

Equivalently, we could say that $\lambda \in \mathbb{C}$ is an eigenvalue if the operator $\mathcal{K} - \lambda\mathcal{I}$ is non-injective. Since, if $\phi_1, \phi_2 \in \mathcal{F}$ are distinct observables with $(\mathcal{K} - \lambda\mathcal{I})\phi_1 = (\mathcal{K} - \lambda\mathcal{I})\phi_2$, then $\mathcal{K}(\phi_1 - \phi_2) = \lambda(\phi_1 - \phi_2)$ and so $\phi_1 - \phi_2$ is an eigenfunction with eigenvalue λ. The set of eigenvalues is referred to as the *point spectrum* of the operator.

For a finite-dimensional operator, injectivity implies surjectivity, and so the point spectrum describes the entire spectrum of the operator. In the infinite-dimensional setting, it is possible for $\mathcal{K} - \lambda\mathcal{I}$ to be injective but not surjective, in which case λ is said to belong to either the *continuous spectrum* or the *residual spectrum*, depending on whether the range is dense in the codomain. Indeed, the spectrum of the Koopman operator may include continuous and residual parts even for seemingly simple dynamics, e.g. the map $f(x) = 2x \pmod 1$ for $x \in [0, 1)$ as considered in [53, Section 2.2]. Nonetheless, the remainder of this discussion will focus on eigenvalues.

We note that the Koopman operator always has an eigenvalue at $\lambda = 1$, corresponding to a constant eigenfunction, since for an observable $\varphi_\alpha(x) := \alpha$ we have

$$\mathcal{K}\varphi_\alpha(x) = \varphi_\alpha(f(x)) = \alpha = \varphi_\alpha(x).$$

For some systems, e.g. mixing attractors such as the Lorenz system, this is the only eigenvalue of the Koopman operator [4]. In contrast, if there exists any eigenvalue λ with $0 < |\lambda| < 1$, then the point spectrum contains countably many distinct eigenvalues, since if $\mathcal{K}\varphi = \lambda\varphi$ then, for all $k \in \mathbb{N}$,

$$\mathcal{K}\varphi^k = (\mathcal{K}\varphi)^k = (\lambda\varphi)^k = \lambda^k\varphi^k,$$

so that λ^k is an eigenvalue with eigenfunction φ^k. More generally, the product $\lambda_1\lambda_2$ of any two eigenvalues is also an eigenvalue (provided that the associated eigenfunctions have non-disjoint support, so that their product $\varphi_1\varphi_2$ is nonzero), since

$$\mathcal{K}(\varphi_1\varphi_2) = (\mathcal{K}\varphi_1)(\mathcal{K}\varphi_2) = \lambda_1\lambda_2\varphi_1\varphi_2.$$

We further note that complex eigenvalues and eigenfunctions must always occur in conjugate pairs. This can be easily verified since, if $\mathcal{K}\varphi = \lambda\varphi$, then

$$\mathcal{K}(\varphi^*) = (\mathcal{K}\varphi)^* = (\lambda\varphi)^* = \lambda^*\varphi^*,$$

so that φ^* is an eigenfunction with eigenvalue λ^*. As a consequence, it is sufficient to consider real-valued observables, as the conjugate pair may be equivalently represented by a pair of real-valued observables. Specifically, we may rewrite the relation

$$\begin{bmatrix} \mathcal{K}\varphi \\ \mathcal{K}\varphi^* \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda^* \end{bmatrix} \begin{bmatrix} \varphi \\ \varphi^* \end{bmatrix}$$

via the linear change of coordinates

$$\begin{bmatrix} \operatorname{Re}\varphi \\ \operatorname{Im}\varphi \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -i/2 & i/2 \end{bmatrix} \begin{bmatrix} \varphi \\ \varphi^* \end{bmatrix}$$

to obtain a real-valued relation

$$\begin{bmatrix}\mathcal{K}(\operatorname{Re}\varphi)\\ \mathcal{K}(\operatorname{Im}\varphi)\end{bmatrix} = \begin{bmatrix}\operatorname{Re}\lambda & -\operatorname{Im}\lambda\\ \operatorname{Im}\lambda & \operatorname{Re}\lambda\end{bmatrix}\begin{bmatrix}\operatorname{Re}\varphi\\ \operatorname{Im}\varphi\end{bmatrix}.$$

2.1.2 Finite-Dimensional Representations

It is difficult to work with the full Koopman operator in practice, since the function space $\mathcal{F}$ is generally infinite-dimensional. As such, it is common to instead consider the finite-dimensional subspace spanned by some finite basis of real-valued observables $\psi_1, \ldots, \psi_N \in \mathcal{F}$. We refer to a vector of observables $\boldsymbol{\Psi} = [\psi_1 \ \cdots \ \psi_N]^T \in \mathcal{F}^N$ as a *dictionary*, for which function evaluation and the action of the Koopman operator apply elementwise as

$$\boldsymbol{\Psi}(x) := \begin{bmatrix}\psi_1(x)\\ \vdots \\ \psi_N(x)\end{bmatrix} \qquad \text{and} \qquad \mathcal{K}\boldsymbol{\Psi} := \begin{bmatrix}\mathcal{K}\psi_1\\ \vdots \\ \mathcal{K}\psi_N\end{bmatrix}$$

respectively. We denote the subspace spanned by its entries as

$$\mathcal{F}_{\boldsymbol{\Psi}} := \left\{\phi \in \mathcal{F} : (\exists \mathbf{v} \in \mathbb{R}^N)[\phi(x) = \mathbf{v}^T\boldsymbol{\Psi}(x), \forall x \in \mathcal{X}]\right\}.$$

A key property for obtaining an exact finite representation is to identify a dictionary whose span is invariant to the Koopman operator in the following sense.

Definition 2.4. *A subspace $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ is $\mathcal{K}$-**invariant** if $\mathcal{K}\phi \in \tilde{\mathcal{F}}$ for all $\phi \in \tilde{\mathcal{F}}$.*

Proposition 2.5. *Given a dictionary $\boldsymbol{\Psi} \in \mathcal{F}^N$, there exists a matrix $A \in \mathbb{R}^{N\times N}$ such that $\mathcal{K}\boldsymbol{\Psi}(x) = A\boldsymbol{\Psi}(x)$ for all $x \in \mathcal{X}$ if and only if $\mathcal{F}_{\boldsymbol{\Psi}}$ is $\mathcal{K}$-invariant.*

Proof: Suppose $\mathcal{F}_{\boldsymbol{\Psi}}$ is $\mathcal{K}$-invariant. Then $\mathcal{K}\psi_j \in \mathcal{F}_{\boldsymbol{\Psi}}$ for each $j \in \{1, \ldots, N\}$, and

so there exists $\mathbf{v}_j \in \mathbb{R}^N$ such that $\mathcal{K}\psi_j(x) = \mathbf{v}_j^T \mathbf{\Psi}(x)$ for all $x \in \mathcal{X}$. Thus, we have

$$\mathcal{K}\mathbf{\Psi}(x) = \begin{bmatrix} \mathcal{K}\psi_1(x) \\ \vdots \\ \mathcal{K}\psi_N(x) \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T \mathbf{\Psi}(x) \\ \vdots \\ \mathbf{v}_N^T \mathbf{\Psi}(x) \end{bmatrix} = \begin{bmatrix} — \mathbf{v}_1^T — \\ \vdots \\ — \mathbf{v}_N^T — \end{bmatrix} \mathbf{\Psi}(x).$$

Conversely, suppose there exists $A \in \mathbb{R}^{N \times N}$ such that $\mathcal{K}\mathbf{\Psi}(x) = A\mathbf{\Psi}(x)$ for all $x \in \mathcal{X}$. Let $\phi \in \mathcal{F}_{\mathbf{\Psi}}$ with $\phi(x) = \mathbf{v}^T \mathbf{\Psi}(x)$. Then

$$\mathcal{K}\phi(x) = \mathbf{v}^T [\mathcal{K}\mathbf{\Psi}(x)] = \mathbf{v}^T A \mathbf{\Psi}(x) = (A^T \mathbf{v})^T \mathbf{\Psi}(x), \qquad \forall x \in \mathcal{X}.$$

Thus, $\mathcal{K}\phi \in \mathcal{F}_{\mathbf{\Psi}}$, and so $\mathcal{F}_{\mathbf{\Psi}}$ is a $\mathcal{K}$-invariant subspace. ■

For a dictionary $\mathbf{\Psi}$ spanning a $\mathcal{K}$-invariant subspace, we may identify eigenfunctions via the left eigenvectors of the corresponding matrix $A \in \mathbb{R}^{N \times N}$. To see this, let $\mathbf{w}$ be a left eigenvector of A^T with eigenvalue λ (so that $\mathbf{w}^T A = \lambda \mathbf{w}^T$). Then

$$\varphi(x) := \mathbf{w}^T \mathbf{\Psi}(x) \tag{2.3}$$

is an eigenfunction with eigenvalue λ, since

$$\mathcal{K}\varphi(x) = \mathbf{w}^T [\mathcal{K}\mathbf{\Psi}(x)] = \mathbf{w}^T A \mathbf{\Psi}(x) = \lambda \mathbf{w}^T \mathbf{\Psi}(x) = \lambda \varphi(x).$$

If the matrix A has a full set of N distinct eigenvectors, then we may decompose $\mathbf{\Psi}(x)$ via these eigenfunctions as described in [58]. Let $\mathbf{v}_j$ and $\mathbf{w}_j$ denote corresponding right and left eigenvectors of A with eigenvalue λ_j, normalized so that $\mathbf{w}_j^T \mathbf{v}_k = \delta_{jk}$. As noted above, the functions $\varphi_j(x) := \mathbf{w}_j^T \mathbf{\Psi}(x)$ are then eigenfunctions with eigenvalue λ_j.

Moreover, we may write

$$\mathbf{\Psi}(x) = \sum_{j=1}^{N} [\mathbf{w}_j^T \mathbf{\Psi}(x)] \mathbf{v}_j = \sum_{j=1}^{N} \varphi_j(x) \mathbf{v}_j.$$

From this, it follows that

$$\mathbf{\Psi}(x_t) = \mathcal{K}^t \mathbf{\Psi}(x_0) = \sum_{j=1}^{N} \mathcal{K}^t \varphi_j(x_0) \mathbf{v}_j = \sum_{j=1}^{N} \lambda_j^t \varphi_j(x_0) \mathbf{v}_j,$$

where x_t denotes the future state t time steps from an initial state x_0. The vectors $\mathbf{v}_j$ are commonly referred to as the *Koopman modes* corresponding to the dictionary $\mathbf{\Psi}$, and the set of triples $\{(\lambda_j, \varphi_j, \mathbf{v}_j)\}_{j=1}^{N}$ as the *Koopman mode decomposition*. For the general case where A does not have a full set of eigenvectors, one may consider generalized eigenvectors and eigenfunctions, as detailed in [45, Section 3.2] for the continuous-time setting.

2.1.3 Data-Driven Approximation

There exist a number of methods for numerically approximating the matrix A of Proposition 2.5 from data. Among these, the current standard is that of *extended mode decomposition* (EDMD) as introduced in [65], which, as the name suggests, is an extension of the *dynamic mode decomposition* (DMD) introduced in [59]. DMD seeks to identify linear dynamics (and the associated Koopman modes) from a single state trajectory, which was generalized in [63] for an unordered set of data pairs. EDMD further generalizes this to identify the dynamics with respect to a specified dictionary of nonlinear observables.

The EDMD algorithm may be described as follows. Given a dictionary $\mathbf{\Psi} \in \mathcal{F}^N$ and a pair of data snapshots

$$\{(x_j, x_j^+)\}_{j=1}^{M} \tag{2.4}$$

satisfying

$$x_j^+ = f(x_j), \qquad \forall j \in \{1, \ldots, M\},$$

we compute a matrix A such that

$$A = \underset{A \in \mathbb{R}^{N \times N}}{\arg\min} \sum_{j=1}^{M} \|\mathbf{\Psi}(x_j^+) - A\mathbf{\Psi}(x_j)\|_2^2, \tag{2.5}$$

where $\|\cdot\|_2$ denotes the standard Euclidean norm.

The solution to (2.5) may be computed in closed form as

$$A = \mathbf{\Psi}_{X^+} \mathbf{\Psi}_X^\dagger,$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse, and $\mathbf{\Psi}_X, \mathbf{\Psi}_{X^+} \in \mathbb{R}^{N \times M}$ are defined respectively as

$$\mathbf{\Psi}_X = \begin{bmatrix} \mathbf{\Psi}(x_1) & \cdots & \mathbf{\Psi}(x_M) \end{bmatrix}, \qquad \mathbf{\Psi}_{X^+} = \begin{bmatrix} \mathbf{\Psi}(x_1^+) & \cdots & \mathbf{\Psi}(x_M^+) \end{bmatrix}.$$

Equivalently, this may be computed without storing the full set of snapshots as

$$A = HG^\dagger,$$

where $G, H \in \mathbb{R}^{N \times N}$ are defined respectively as

$$G = \frac{1}{M} \sum_{j=1}^{M} \mathbf{\Psi}(x_j)\mathbf{\Psi}(x_j)^T, \qquad H = \frac{1}{M} \sum_{j=1}^{M} \mathbf{\Psi}(x_j^+)\mathbf{\Psi}(x_j)^T.$$

If $\mathcal{F}_{\mathbf{\Psi}}$ is a $\mathcal{K}$-invariant subspace, then the optimal value of (2.5) will be zero. Assuming the data set (2.4) is sufficiently rich, then this further implies that we will have $\mathcal{K}\mathbf{\Psi}(x) = A\mathbf{\Psi}(x)$ for all $x \in \mathcal{X}$. If $\mathcal{F}_{\mathbf{\Psi}}$ is not $\mathcal{K}$-invariant, then the optimal value of (2.5) will be

nonzero, and $\mathcal{K}\mathbf{\Psi}(x) \approx A\mathbf{\Psi}(x)$ will only hold approximately. However, it is known that (with a few reasonable assumptions) the EDMD approximation converges in the strong operator topology to the true Koopman operator in the limit as the number of snapshots M and the size of the dictionary N go to infinity [36].

For numerical computation, it may be helpful to implement incorporate regularization, which is commonly done via a truncated singular value decomposition. This is closely related to Tikhonov regularization [25], in which a regularization term on the Frobenius norm is added to (2.5). Regularization of other norms may also be considered, such as the L_1-norm to promote sparsity [32], or the $L_{2,1}$-norm to encourage A to have a small number of nonzero columns [64].

2.2 Actuated Systems

To make use of the Koopman operator in control problems, we must extend the above theory to systems with inputs. Denoting the input as $u \in \mathcal{U}$, the dynamics are described by a map $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ such that

$$x^+ = f(x, u). \tag{2.6}$$

As with the state, we may consider the input set $\mathcal{U}$ to consist of abstract, non-numeric elements. There are multiple ways of representing such systems in the Koopman framework, which we discuss now.

2.2.1 Extended State Space

This section discusses the type of representation formalized in [35]. For the system (2.6), we consider the associated autonomous system on an extended state space $\mathcal{X} \times \ell(\mathcal{U})$,

where $\ell(\mathcal{U})$ denotes the space of all infinite input sequences. The extended state

$$\chi := (x,\, \boldsymbol{\mu}^\infty)\,, \qquad \text{where} \qquad \boldsymbol{\mu}^\infty := (u_t)_{t=0}^\infty = (u_0,\, u_1,\, \dots),$$

then evolves according to the map $F : \mathcal{X} \times \ell(\mathcal{U}) \to \mathcal{X} \times \ell(\mathcal{U})$ defined as

$$\chi^+ = F(\chi) := (f(x, \boldsymbol{\mu}^\infty(0)),\, S\boldsymbol{\mu}^\infty)\,, \tag{2.7}$$

where $\boldsymbol{\mu}^\infty(j)$ denotes the jth element of $\boldsymbol{\mu}^\infty$, and S is the left shift operator such that $(S\boldsymbol{\mu}^\infty)(j) = \boldsymbol{\mu}^\infty(j+1)$.

As an autonomous system, the associated Koopman operator is well-defined by (2.2) for a function space $\mathcal{F}_\infty$ of observables on the extended state space, i.e. $\phi : \mathcal{X} \times \ell(\mathcal{U}) \to \mathbb{R}$. For a dictionary $\boldsymbol{\Phi} \in \mathcal{F}_\infty^{N_\phi}$ spanning a Koopman-invariant subspace, Proposition 2.5 guarantees the existence of a matrix $\mathcal{A} \in \mathbb{R}^{N_\phi \times N_\phi}$ such that

$$\boldsymbol{\Phi}(\chi^+) = \mathcal{K}\boldsymbol{\Phi}(\chi) = \mathcal{A}\boldsymbol{\Phi}(\chi), \qquad \forall \chi \in \mathcal{X} \times \ell(\mathcal{U}).$$

Note that, although $\boldsymbol{\Phi}(\chi)$ is finite-dimensional, it generally cannot be computed in finite time since it depends on the infinite-dimensional extended state $\chi = (x, \boldsymbol{\mu}^\infty)$. However, for the purpose of control, we are only interested in predicting functions of the original state x, which by causality depend only on the first input u of the infinite sequence $\boldsymbol{\mu}^\infty$.

For the following, we suppose the input is given by a vector $\mathbf{u} = [u_1 \ \cdots \ u_m]^T \in \mathbb{R}^m$. For a dictionary $\boldsymbol{\Phi} \in \mathcal{F}_\infty^{N+m}$ of the form

$$\boldsymbol{\Phi}(\chi) = \boldsymbol{\Phi}(x, \boldsymbol{\mu}^\infty) = \begin{bmatrix} \boldsymbol{\Psi}(x_t) \\ \boldsymbol{\mu}^\infty(0) \end{bmatrix},$$

where $\mathbf{\Psi} \in \mathcal{F}^N$, we then have that

$$\mathcal{K}\mathbf{\Phi}(\chi) = \mathbf{\Phi}(f(x, \boldsymbol{\mu}^\infty(0)),\, S\boldsymbol{\mu}^\infty) = \begin{bmatrix} \mathbf{\Psi}(f(x, \boldsymbol{\mu}^\infty(0))) \\ \boldsymbol{\mu}^\infty(1) \end{bmatrix}.$$

Since $\mathbf{\Phi}$ does not contain observables that depend on future inputs, then $\mathcal{K}\phi_j \notin \mathcal{F}_{\mathbf{\Phi}}$ for $j \in \{N+1, \ldots, N+m\}$, and so $\mathcal{F}_{\mathbf{\Phi}}$ is not $\mathcal{K}$-invariant. However, this is not necessarily a problem since we are not interested in being able to predict future control inputs. We may still hope to have $\mathcal{K}\phi_j \in \mathcal{F}_{\mathbf{\Phi}},\ j \in \{1, \ldots, N\}$, in which case there exist matrices $A \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times m}$ such that

$$\mathbf{\Psi}(x^+) = A\mathbf{\Psi}(x) + B\mathbf{u}. \tag{2.8}$$

Linear predictors of this form are immediately amenable to linear control methods such as model-predictive control.

One may also obtain a more general bilinear form via a dictionary $\mathbf{\Phi} : \mathcal{X} \times \ell(\mathcal{U}) \to \mathbb{R}^{N+Nm+m}$ of the form

$$\mathbf{\Phi}(\chi) = \begin{bmatrix} \mathbf{\Psi}(x) \\ \boldsymbol{\mu}^\infty(0) \otimes \mathbf{\Psi}(x) \\ \boldsymbol{\mu}^\infty(0) \end{bmatrix},$$

such as considered in [12], where $\otimes$ denotes the Kronecker product such that $\mathbf{u} \otimes \mathbf{\Psi}(x) = [u_1\mathbf{\Psi}(x)^T \ \cdots \ u_m\mathbf{\Psi}(x)^T]^T$. If this dictionary has $\mathcal{K}\phi_j \in \mathcal{F}_{\mathbf{\Phi}},\ j \in \{1, \ldots, N\}$ then there will exist matrices $A_0, \ldots, A_m \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times m}$ such that

$$\mathbf{\Psi}(x^+) = A_0\mathbf{\Psi}(x) + \sum_{j=1}^{m} u_j A_j \mathbf{\Psi}(x) + B\mathbf{u}. \tag{2.9}$$

This type of representation is particularly suited control-affine systems, as discussed in [62].

2.2.2 Input-Parameterization

An alternative representation is that of an input-parameterized Koopman operator. In this setting, we view the system (2.6) as a family of autonomous subsystems defined for each constant input $u \in \mathcal{U}$ by

$$x^+ = f_u(x) := f(x, u).$$

Each constant-input subsystem is autonomous, so that (2.2) defines an associated Koopman operator $\mathcal{K}(u) : \mathcal{F} \to \mathcal{F}$ by

$$\mathcal{K}(u)\phi := \phi \circ f_u, \qquad \forall \phi \in \mathcal{F}.$$

The collection of operators $\{\mathcal{K}(u)\}_{u\in\mathcal{U}}$ then constitutes the "Koopman control family" defined in [27].

In this context, we discuss Koopman-invariance as a property of the Koopman control family as a whole, rather than of an individual operator $\mathcal{K}(u)$.

Definition 2.6. *A subspace $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ is **Koopman-invariant** if it is $\mathcal{K}(u)$-invariant for all $u \in \mathcal{U}$, i.e. if*

$$\mathcal{K}(u)\phi \in \tilde{\mathcal{F}}, \qquad \forall \phi \in \tilde{\mathcal{F}},\ \forall u \in \mathcal{U}.$$

For a dictionary $\mathbf{\Psi} \in \mathcal{F}^N$ such that $\mathcal{F}_{\mathbf{\Psi}}$ is Koopman-invariant, Proposition 2.5 provides us with a family of matrices $A(u) \in \mathbb{F}^{N\times N}, u \in \mathcal{U}$ such that

$$\mathcal{K}(u)\mathbf{\Psi}(x) = A(u)\mathbf{\Psi}(x), \qquad \forall x \in \mathcal{X},\ \forall u \in \mathcal{U}. \tag{2.10}$$

We remark that this seemingly small change leads to significant consequences. For example, we recall that the relation $\mathcal{K}\mathbf{\Psi}(x) = A\mathbf{\Psi}(x)$ for an autonomous system allows us

to identify eigenfunctions via the left eigenvectors of A. However, we do not get this nice property for (2.10), since there is generally no relation between the left eigenvectors of $A(u_1), A[u_2]$ for distinct inputs u_1, u_2. Even in general, a notion of a "joint eigenfunction" with $\mathcal{K}(u)\varphi = \lambda(u)\varphi$ for all $u \in \mathcal{U}$ would be highly restrictive, since then

$$\mathcal{K}(u_1)\mathcal{K}(u_2)\varphi = \mathcal{K}(u_1)\lambda(u_2)\varphi = \lambda(u_1)\lambda(u_2)\varphi = \mathcal{K}(u_2)\lambda(u_1)\varphi = \mathcal{K}(u_2)\mathcal{K}(u_1)\varphi,$$

implying that φ must be invariant to permutations of the input sequence.

We further remark that, although (2.10) most naturally describes a switched system representation, it is also general enough to include linear and bilinear representations. Specifically, by considering a dictionary augmented with the constant function $\mathbb{1}(x) := 1$, the linear representation (2.8) is expressed equivalently in the form of (2.10) as

$$\begin{bmatrix} \mathbf{\Psi}(x^+) \\ \mathbb{1}(x^+) \end{bmatrix} = \begin{bmatrix} A & B\mathbf{u} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}(x) \\ \mathbb{1}(x) \end{bmatrix}.$$

Similarly, the bilinear representation (2.9) is expressed equivalently as

$$\begin{bmatrix} \mathbf{\Psi}(x^+) \\ \mathbb{1}(x^+) \end{bmatrix} = \begin{bmatrix} A_0 + \sum_{j=1}^{m} u_j A_j & B\mathbf{u} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}(x) \\ \mathbb{1}(x) \end{bmatrix}.$$

In fact, the input-parameterized Koopman operator has equivalent descriptive power to the Koopman operator for the extended state space system (2.7), as studied in [28].

2.3 Output Representations

Given a dictionary $\mathbf{\Psi}$ spanning a Koopman-invariant subspace, we are able to accurately predict future values of observables within that subspace as linear combinations

of the lifted state $\mathbf{\Psi}(x)$. However, the utility of this relies on the underlying subspace being "useful" in some sense. For example, the subspace of constant functions is trivially Koopman-invariant for any system, but the only "prediction" this allows us to make is that constant functions will remain constant.

If the state has a vector representation $\mathbf{x} = [x_1 \ \cdots \ x_n]^T \in \mathbb{R}^n$, then it is reasonable to consider state-inclusive dictionaries $\mathbf{\Psi}$, i.e. those which (after a possible reordering) have $\psi_j(\mathbf{x}) = x_j$ for $j \in \{1, \ldots, n\}$. A state-inclusive dictionary spanning a Koopman-invariant subspace enables linear prediction of future states, which is useful in certain applications such as forecasting or state regulation.

Depending on desired application, the requirement of a state-inclusive dictionary may be stronger than necessary (if we only need to predict certain components of the state) or weaker than necessary (if we need to predict observables other than the state-components). Moreover, the states $x \in \mathcal{X}$ may in general be abstract objects that lack a natural numeric representation, in which case this notion of state-inclusivity is ill-defined.

In the most general setting, we may augment the system (2.6) with an output mapping $\mathbf{g} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^{n_y}$, denoting the functions that we are interested in predicting. Thus, we consider a dynamical system with state $x \in \mathcal{X}$, input $u \in \mathcal{U}$, and output $\mathbf{y} \in \mathbb{R}^{n_y}$, that are related to each other as follows.

$$x^+ = f(x, u), \tag{2.11a}$$

$$\mathbf{y} = \mathbf{g}(x, u). \tag{2.11b}$$

Note that, although we still consider the state and input may be non-numeric in general, the output is a user-defined vector of numeric measurements. This is difference is reflected in the boldface notation for $\mathbf{y}$.

Our goal is then to identify a linear representation of the system with the following

form.

Definition 2.7. *Given a dictionary $\boldsymbol{\Psi} \in \mathcal{F}^N$ and matrix-valued functions $A : \mathcal{U} \to \mathbb{R}^{N\times N}$, $C : \mathcal{U} \to \mathbb{R}^{n_y \times N}$, we say that $(\boldsymbol{\Psi}, A, C)$ is an **exact Koopman representation** for (2.11) if*

$$\boldsymbol{\Psi}(f(x,u)) = A(u)\boldsymbol{\Psi}(x), \tag{2.12a}$$

$$\mathbf{g}(x,u) = C(u)\boldsymbol{\Psi}(x), \tag{2.12b}$$

for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$.

To obtain such a representation, we require that subspace spanned by $\boldsymbol{\Psi}$ is both Koopman-invariant in the sense of Definition 2.6 and *output-inclusive* in the following sense.

Definition 2.8. *A subspace $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ is **output-inclusive** for the system (2.11) if $g_{u,j} \in \tilde{\mathcal{F}}$ for all $u \in \mathcal{U}$ and $j \in \{1, \dots, n_y\}$, where $g_{u,j}$ denotes the jth component of the map $\mathbf{g}_u(x) := \mathbf{g}(x,u)$.*

Proposition 2.9. *A dictionary $\boldsymbol{\Psi} \in \mathcal{F}^N$ admits an exact Koopman representation (i.e. there exist $A : \mathcal{U} \to \mathbb{R}^{N\times N}$ and $C : \mathcal{U} \to \mathbb{R}^{n_y \times N}$ such that $(\boldsymbol{\Psi}, A, C)$ is an exact Koopman representation) if and only if $\mathcal{F}_{\boldsymbol{\Psi}}$ is both Koopman-invariant and output-inclusive.*

The proof of this statement very closely follows that of Proposition 2.5.

Recall that, for any system, we can always identify a finite-dimensional Koopman-invariant subspace (the space of constant functions), but this subspace is not output-inclusive. Similarly, for a system with a finite input set $\mathcal{U} = \{u_1, \dots, u_{n_u}\}$ the space spanned by $\boldsymbol{\Psi}(x) = [\mathbf{g}_{u_1}(x)^T \ \cdots \ \mathbf{g}_{u_{n_u}}(x)^T]^T$ is output-inclusive, but in general is not Koopman-invariant. It is also always possible to identify a subspace satisfying both properties, but any such subspace will generally be infinite-dimensional.

Proposition 2.10. *Let*

$$\mathcal{G}_0 := \{g_{u,j} : u \in \mathcal{U},\, j \in \{1,\dots,n_y\}\},$$
$$\mathcal{G}_{k+1} := \{\mathcal{K}(u)\phi : u \in \mathcal{U},\, \phi \in \mathcal{G}_k\}, \qquad \forall k \geq 0.$$

Then the subspace $\mathcal{G}_\infty := \operatorname{span}\bigcup_{k=0}^{\infty}\mathcal{G}_k$ *is both Koopman-invariant and output-inclusive. Moreover, it is the smallest such subspace, in the sense that if* $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ *is a Koopman-invariant and output-inclusive subspace, then* $\mathcal{G}_\infty \subseteq \tilde{\mathcal{F}}$.

Proof: It is clear that $\mathcal{G}_\infty$ is output-inclusive, since for any $u \in \mathcal{U},\ j \in \{1,\dots,n_u\}$ we have $g_{u,j} \in \mathcal{G}_0 \subseteq \mathcal{G}_\infty$. Now, let $\psi \in \mathcal{G}_\infty$. Thus, we may write $\psi = \sum_{j=1}^{\infty}\alpha_j\phi_j$ for some $\alpha_j \in \mathbb{R},\ \phi_j \in \bigcup_{k=0}^{\infty}\mathcal{G}_k$. For each j, there then exists k such that $\phi_j \in \mathcal{G}_k$, and thus $\mathcal{K}(u)\phi_j \in \mathcal{G}_{k+1}$ for any $u \in \mathcal{U}$. Therefore, we have $\mathcal{K}(u)\psi = \sum_{j=1}^{\infty}\alpha_j\mathcal{K}(u)\phi_j \in \operatorname{span}\bigcup_{k=0}^{\infty}\mathcal{G}_k = \mathcal{G}_\infty$, so that $\mathcal{G}_\infty$ is Koopman-invariant.

We now show that $\mathcal{G}_\infty$ is the smallest such subspace. Let $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ be a Koopman-invariant and output-inclusive subspace. Since $\tilde{\mathcal{F}}$ is output-inclusive, then $g_{u,j} \in \tilde{\mathcal{F}}$ for all $u \in \mathcal{U},\ j \in \{1,\dots,n_y\}$, and so $\mathcal{G}_0 \subseteq \tilde{\mathcal{F}}$. Suppose now, for some $k \geq 0$, that $\mathcal{G}_k \subseteq \tilde{\mathcal{F}}$. Then, since $\tilde{\mathcal{F}}$ is Koopman-invariant, we have that $\mathcal{K}(u)\phi \in \tilde{\mathcal{F}}$ for all $u \in \mathcal{U}$ and $\phi \in \mathcal{G}_k$, and so $\mathcal{G}_{k+1} \subseteq \tilde{\mathcal{F}}$. By induction, we then have $\mathcal{G}_k \subseteq \tilde{\mathcal{F}}$ for all $k \geq 0$. Since $\tilde{\mathcal{F}}$ is a subspace, then it contains all linear combinations of the observables in the sets $\mathcal{G}_k$, and so we conclude that $\mathcal{G}_\infty = \operatorname{span}\bigcup_{k=0}^{\infty}\mathcal{G}_k \subseteq \tilde{\mathcal{F}}$. ■

In general, the infinite union of the sets $\mathcal{G}_k$ consists of infinitely many linearly independent observables, so that $\mathcal{G}_\infty$ is infinite-dimensional. As a consequence, there generally does not exist a finite-dimensional subspace that is both Koopman-invariant and output-inclusive. In practice, we must then work with dictionaries that yield only approximate Koopman representations, i.e. representations for which at least one of (2.12a) and (2.12b) is not satisfied.

The matrices $A(u)$ and $C(u)$ can be approximately learned from data via a slight modification to the EDMD algorithm described in Section 2.1.3. Suppose we have a dictionary $\mathbf{\Psi} \in \mathcal{F}^N$ and a set of data snapshots

$$\{(x_j, x_j^+, u_j, \mathbf{y}_j)\}_{j=1}^M \tag{2.13}$$

satisfying

$$x_j^+ = f(x_j, u_j), \quad \mathbf{y}_j = \mathbf{g}(x_j, u_j), \qquad \forall j \in \{1, \dots, M\}\,.$$

We partition the full set of snapshots into constant input sets $\{(x_j, x_j^+, u_j, \mathbf{y}_j)\}_{j \in \mathcal{J}(u)}$, where $\mathcal{J}(u) := \{j \in \{1, \dots, M\} : u_j = u\}$. Each partition then corresponds to an autonomous subsystem, for which we may apply the standard EDMD algorithm (2.5).

The matrices $C(u)$ may be learned in the same way, but with the outputs $\mathbf{y}_j$ in place of the lifted states $\mathbf{\Psi}(x_j^+)$. However, since we consider the output functions to be user-specified, it may be known that certain functions depend only on the state and not on the input. In this case, we may partition the output $y \in \mathbb{R}^{n_y}$ (after a possible reordering) as

$$\mathbf{y} = \begin{bmatrix} \tilde{\mathbf{y}} \\ \bar{\mathbf{y}} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{g}}(x, u) \\ \bar{\mathbf{g}}(x) \end{bmatrix},$$

where $\tilde{\mathbf{y}} \in \mathbb{R}^{n_{\tilde{y}}}$, $\bar{\mathbf{y}} \in \mathbb{R}^{n_{\bar{y}}}$ with $n_{\tilde{y}} + n_{\bar{y}} = n_y$. We then approximate $\tilde{\mathbf{y}}$ via input-dependent matrices learned from the partitioned snapshots, and approximate $\bar{\mathbf{y}}$ via a single matrix learned from the full set of snapshots.

We thus obtain the matrices $A(u)$ and $C(u)$ for systems with inputs and outputs as

$$A(u) = \underset{A \in \mathbb{R}^{N \times N}}{\arg\min} \sum_{j \in \mathcal{J}(u)} \|\boldsymbol{\Psi}(x_j^+) - A\boldsymbol{\Psi}(x_j)\|_2^2, \tag{2.14a}$$

$$C(u) = \begin{bmatrix} \tilde{C}(u) \\ \\ \bar{C} \end{bmatrix} = \begin{bmatrix} \underset{\tilde{C} \in \mathbb{R}^{n_{\tilde{y}} \times N}}{\arg\min} \sum_{j \in \mathcal{J}(u)} \|\tilde{\mathbf{y}}_j - \tilde{C}\boldsymbol{\Psi}(x_j)\|_2^2 \\ \\ \underset{\bar{C} \in \mathbb{R}^{n_{\bar{y}} \times N}}{\arg\min} \sum_{j=1}^{M} \|\bar{\mathbf{y}}_j - \bar{C}\boldsymbol{\Psi}(x_j)\|_2^2 \end{bmatrix}. \tag{2.14b}$$

Chapter 3

Entity-Based Systems

This chapter discusses a novel class of "entity-based" systems as introduced in [9] and expanded upon in [10]. We first describe the concepts in a general sense, and then we define types of observables that allow them to be represented and modeled via the Koopman operator.

3.1 Motivating Examples

Before defining the concept of an entity-based system, it is helpful to introduce concrete examples to serve as illustrative examples. These example systems will also be used to demonstrate numerical results through the remainder of this work.

3.1.1 MultiPong

We first consider the *MultiPong* system as introduced in [10], which a relatively simple game implemented in Julia. It consists of a 1×1 box containing some number of "balls" and "paddles". Balls are denoted by points with associated velocity vectors, and paddles are denoted by vertical line segments with length 0.1. The balls move according

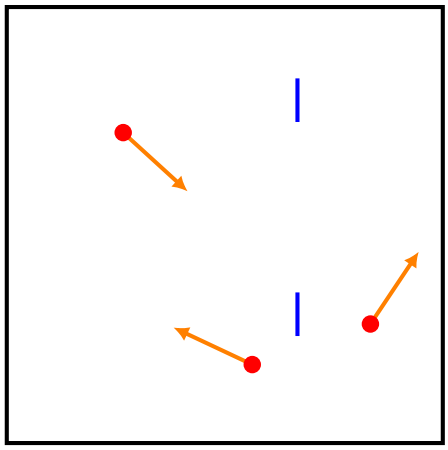

Figure 3.1: A sample state from a MultiPong game with three balls and two paddles.

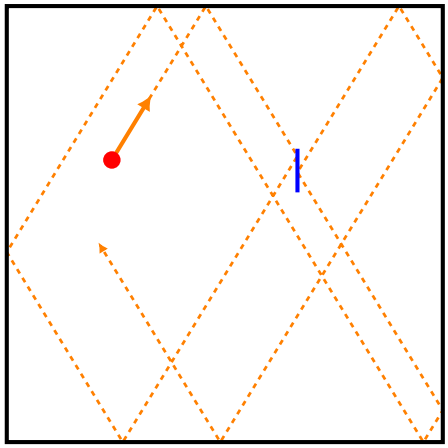

Figure 3.2: An example game of MultiPong with one ball and one paddle. A depiction of the future trajectory of the ball illustrates how it collides with the walls and with the paddle.

to their velocity, and collide elastically with the walls of the box and with the paddles. The paddles move up and down according to a chosen input. Figure 3.1 illustrates an example state of the system with multiple balls and paddles, and Figure 3.2 illustrates how the ball moves within in the box.

Paddles move up and down between twelve fixed y-positions between 0.05 and 0.95

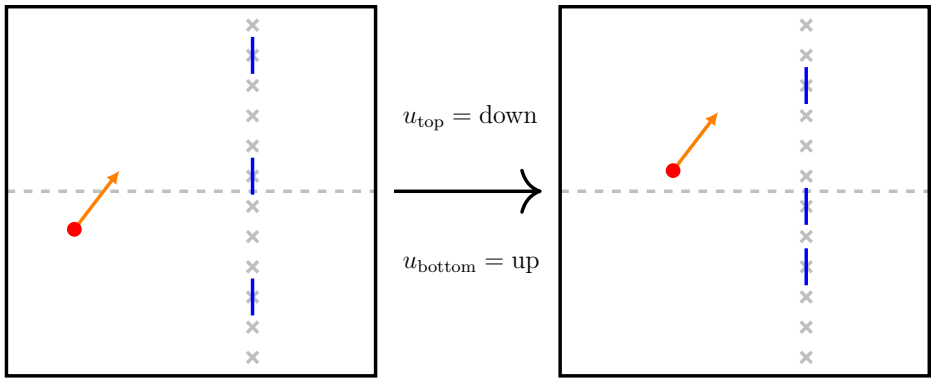

Figure 3.3: An illustration of the control input for MultiPong. The paddles move among the fixed coordinates marked by the crosses. Paddles above the dashed line move according to the input u_{top}, while paddles below the dashed line move according the input u_{bottom}.

Inputs to the system are in the form of pairs

$$u = (u_{\text{top}}, u_{\text{bottom}}) \in \{\text{up}, \text{down}, \text{stay}\}^2.$$

The chosen u_{top} and u_{bottom} are broadcast to paddles in the top and bottom halves of the box respectively, as illustrated in Figure 3.3. Paddles that receive an input of 'up' or 'down' move in the corresponding direction (unless they are already at the topmost or bottommost position respectively). Paddles that receive the input 'stay' do not move.

One property of the MultiPong system that will be helpful for our analysis is that we can specify the initial state in such a way that only a finite number of states are reachable. This will guarantee that we are able to identify and exact Koopman representation. Letting $m_x, m_y \in \mathbb{N}$, we choose initial positions and velocities of each ball as

$$b_x = \frac{1}{2m_x}, \qquad b_y \in \left\{\frac{2k-1}{2m_y}\right\}_{k=1}^{m_y}, \qquad b_{vx} = \frac{1}{m_x}, \qquad b_{vy} \in \left\{-\frac{1}{m_y}, \frac{1}{m_y}\right\}. \tag{3.1}$$

Without paddle collisions, the positions reachable by the ball then form an $m_x \times m_y$ grid as shown in Figure 3.4. This grid of reachable states is preserved if the paddles are

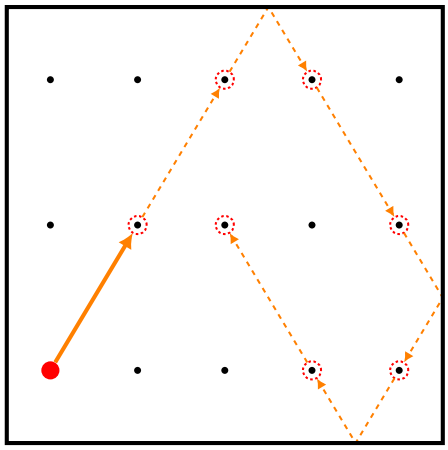

Figure 3.4: An illustration of the trajectory of a ball initialized according to (3.1) with $m_x = 5$ and $m_y = 3$. The future positions of the ball remain confined to the 5×3 grid of points as shown.

placed at x-positions exactly halfway between consecutive points on the grid. We do this by choosing the position as

$$p_x = \frac{\text{round}(2m_x/3)}{m_x},$$

which places the paddles at the nearest such position to $x = 2/3$.

Restricted to this finite set of states, there are only finitely many linearly independent observables, and so there exists a finite-dimensional subspace that is both Koopman-invariant and output-inclusive. By introducing a tunable perturbation to the initial state, we can tune the proximity of the reachable state set to the finite state set, thus tuning the accuracy of Koopman model we are able to obtain. We do this by choosing a small perturbation scale $\delta > 0$, and then replacing the initial ball velocities in (3.1) with

$$b_{vx} = \frac{\alpha}{m_x}, \qquad b_{vy} \in \left\{ -\frac{\beta}{m_y}, \frac{\beta}{m_y} \right\}, \tag{3.2}$$

for α, β chosen uniformly from the interval $[1-\delta, 1+\delta]$. Examples of the perturbed ball trajectories for multiple values of δ are illustrated in Figure 3.5.

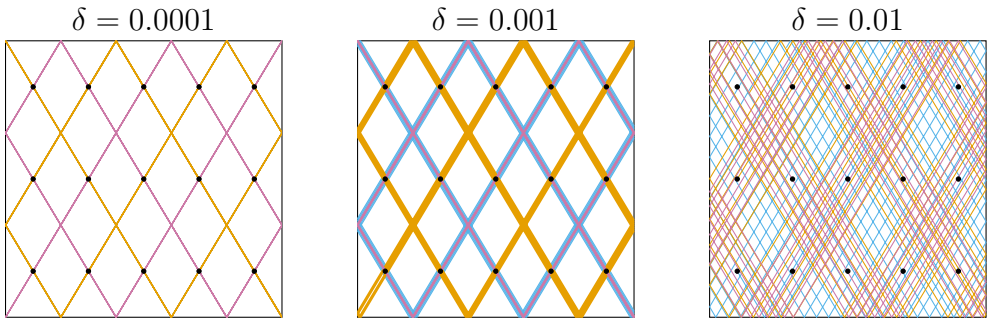

Figure 3.5: A visualization of sample trajectories of balls with perturbed initial velocities as in (3.2).

3.1.2 Assault

We next consider the Atari 2600 game *Assault*, shown in Figure 3.6. In this game, the player controls a ship near the bottom of the screen, and wishes to fire at and defeat enemies that appear above them. At each moment, the player may take one of the following six action: move left, move right, fire left, fire right, fire up, or do nothing. If the player fires at and hits an enemy, then that enemy is defeated and the player scores a fixed number of points. Conversely, if the player is hit by enemy fire, then they are killed and lose a life. A gauge in the bottom right also increases when the player fires, and slowly decreases over time when the player does not. This serves to limit how often the player may fire, since they lose a life if they fire when the gauge is already full. The game ends when the player loses all four of their lives.

3.2 Entity-Based Representations

We now define the concept of an *entity-based system*, as introduced in [9]. This term refers to dynamical system of the form (2.11) whose state x consists of some (possibly varying) number of components, which we shall refer to as "entities". Each such entity has its own individual state, and multiple entities may be organized into distinct classes

Figure 3.6: An example screen of the game Assault.

for which we assume the following:

- Each entity in class i has an individual state that may be represented by a vector in $\mathbb{R}^{m_i}$;
- Entities within the same class have similar behavior, and thus may be treated anonymously;
- The number of entity classes is fixed, although the number of entities within each class may vary.

Given these assumptions, it is natural to associate each class of entities with a set-valued mapping $\mathcal{E}_i : \mathcal{X} \rightrightarrows \mathbb{R}^{m_i}$, with the understanding that the set $\mathcal{E}_i(x)$ includes the states of all entities of class i.

Formally, we define an entity-based system as follows.

Definition 3.1. *An **entity-based representation** for the system* (2.11) *is a mapping of the form*

$$\mathscr{E}(x) := (\mathcal{E}_1(x), \ldots, \mathcal{E}_{n_e}(x)),$$

for set-valued mappings $\mathcal{E}_i : \mathcal{X} \rightrightarrows \mathbb{R}^{m_i}, i \in \{1, \ldots, n_e\}$. *We denote the codomain of* $\mathscr{E}$ *by* $\mathscr{X} := 2^{\mathbb{R}^{m_1}} \times \cdots \times 2^{\mathbb{R}^{m_{n_e}}}$. *An* ***entity-based system*** *is then a system with an entity-based representation* $\mathscr{E}$ *such that there exist mappings* $\mathscr{F} : \mathscr{X} \times \mathcal{U} \to \mathscr{X}$ *and* $\mathscr{G} : \mathscr{X} \times \mathcal{U} \to \mathbb{R}^{n_y}$ *such that*

$$\mathscr{E}(f(x, u)) = \mathscr{F}(\mathscr{E}(x), u), \tag{3.3a}$$

$$\mathbf{g}(x, u) = \mathscr{G}(\mathscr{E}(x), u), \tag{3.3b}$$

for all $x \in \mathcal{X}$ *and* $u \in \mathcal{U}$.

More generally, for systems where multiple entities may have identical states, the mappings $\mathcal{E}_i$ may be viewed as multiset-valued, i.e. with each $\mathcal{E}_i(x)$ being a set that allows for repeated elements. In either case, we assume that each $\mathcal{E}_i(x)$ contains only finitely entities.

As with the Koopman-based representations described by Definition 2.7, both conditions (3.3a) and (3.3b) are necessary in order to ensure that we have a "useful" representation of the system (2.11). Figure 3.7 illustrates how both entity-based representations and Koopman-based representations capture the same state-input-output relations of the underlying system.

We shall now this framework in relation to the previously introduced system. Specifically, we describe limitations imposed by a traditional state-space representation, and then propose an alternative representation via entities.

3.2.1 Entities for MultiPong

Each ball has an individual state $\mathbf{b} = [b_x \ b_y \ b_{vx} \ b_{vy}]^T \in \mathbb{R}^4$, where (b_x, b_y) denotes its (x, y)-position and (b_{vx}, b_{vy}) denotes its (x, y)-velocity. Each paddle has an individual

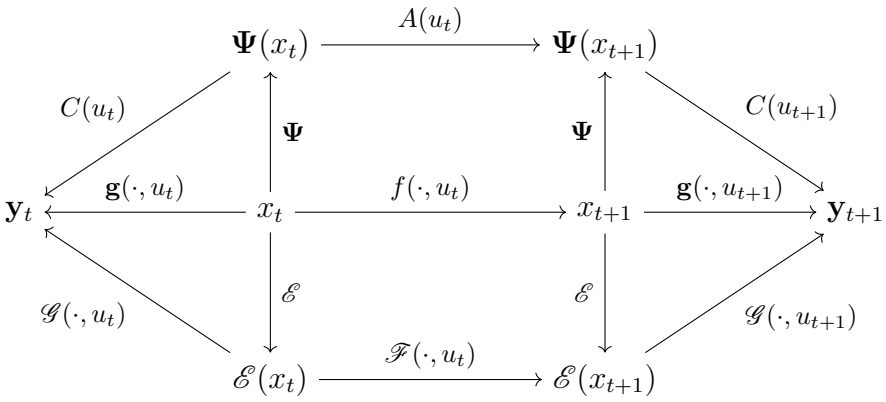

Figure 3.7: A diagram illustrating the relations between states, inputs, and outputs for the exact Koopman representations described by Definition 2.7, entity-based representations described by Definition 3.1, and the underlying dynamical system (2.11).

state $\mathbf{p} = [p_x \ p_y]^T \in \mathbb{R}^2$, denoting the (x, y)-position of its center. A state-space representation for the system could be constructed via a concatenation of these individual states. For example, consider the system state shown in Figure 3.1. One could label the ball states as

$$\mathbf{b}_1 = \begin{bmatrix} 0.267 \\ 0.712 \\ 0.148 \\ -0.135 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 0.563 \\ 0.179 \\ -0.181 \\ 0.085 \end{bmatrix}, \quad \mathbf{b}_3 = \begin{bmatrix} 0.834 \\ 0.273 \\ 0.111 \\ 0.166 \end{bmatrix},$$

and the paddle states as

$$\mathbf{p}_1 = \begin{bmatrix} 0.7 \\ 0.295 \end{bmatrix}, \qquad \mathbf{p}_2 = \begin{bmatrix} 0.7 \\ 0.786 \end{bmatrix}.$$

A concatenation of these individual entity states would yield a vector representation as

$$\mathbf{x} = \begin{bmatrix} \mathbf{b}_1^T & \mathbf{b}_2^T & \mathbf{b}_3^T & \mathbf{p}_1^T & \mathbf{p}_2^T \end{bmatrix}^T \in \mathbb{R}^{16}. \tag{3.4}$$

We note two key shortcomings with this vector representation. First, this repre-

sentation contains inherent redundancies due to the arbitrary labeling of the states. For example, the state vector in (3.4) is numerically distinct from the vector $\mathbf{x} = [\mathbf{b}_3^T\ \mathbf{b}_1^T\ \mathbf{b}_2^T\ \mathbf{p}_2^T\ \mathbf{p}_1^T]^T$, even though both describe the same state shown in Figure 3.1. Second, the dimension of the state vector $\mathbf{x}$ is dependent on the number of entities. For example, a state with three balls and two paddles would have $\mathbf{x} \in \mathbb{R}^{16}$ as in (3.4). In contrast, a state with, say, two balls and one paddle would have $\mathbf{x} = [\mathbf{b}_1^T\ \mathbf{b}_2^T\ \mathbf{p}_1^T]^T \in \mathbb{R}^{10}$. This may be particularly problematic if the number of entities may vary over a single trajectory. However, even if this is not the case, a representation that is independent of the number of entities may be preferable to enable the same Koopman dictionary and representation to be reused for different numbers of entities.

In contrast, consider an entity-based representation with two entity classes, where $\mathcal{E}_1(x) \subseteq \mathbb{R}^4$ contains the states of the balls and $\mathcal{E}_2(x) \subseteq \mathbb{R}^2$ contains the states of the paddles. For the system state shown in Figure 3.1, we then have

$$\mathscr{E}(x) = (\mathcal{E}_1(x), \mathcal{E}_2(x)) = (\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\},\ \{\mathbf{p}_1, \mathbf{p}_2\}) \in 2^{\mathbb{R}^4} \times 2^{\mathbb{R}^2}.$$

We note that relabeling the states has no effect on representation $\mathscr{E}(x)$, since the sets are unordered and so e.g. $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\} = \{\mathbf{b}_3, \mathbf{b}_1, \mathbf{b}_2\}$. Moreover, we always have $\mathscr{E}(x) \in 2^{\mathbb{R}^4} \times 2^{\mathbb{R}^2}$ regardless of how many balls and paddles are present.

3.2.2 Entities for Assault

For Assault, the description of the state is more complicated. In principle, the state is described by the 1024-bit memory of the Atari 2600 console, and so could be represented by a vector in $\mathbb{R}^{1024}$. However, the system memory is not directly accessible, and would be difficult to interpret even if it were. Instead, we observe the game screen as a proxy for the state, which consists of a 160-by-210 grid of pixels with three color channels each.

Table 3.1: Summary of the five entity classes for Assault.

i	Entity class
1	Player ship
2	Player fire
3	Enemy ships
4	Enemy fire
5	Attack gauge

The screen observation may then be represented by an array in $\mathbb{R}^{3\times160\times210}$. Rather than working directly with this large array, we would like to identify and extract a smaller set of meaningful features.

Various on-screen objects may be described as entities belonging to a number of distinct classes as listed in Table 3.1 and annotated in Figure 3.8. Since the number of objects present changes within a single episode of the game, it is not feasible to concatenate them into a fixed-length state vector. However, this is not an issue for our entity-based representation. A similar description of Atari 2600 games via classes of objects is described in [50], which may be viewed as a specific type of our more general entity-based framework.

We remark that we may describe the individual pixels of objects in each class as separate entities, eliminating the need to extract separate "blobs" of connected pixels. For example, consider the three enemy ships belonging to entity class 3 in Figure 3.8. It is straightforward to identify the 165 individual pixels that make up these ships, and we define the set $\mathcal{E}_3(x)$ as the set of these pixels' coordinates. This saves us the work of determining how many ships are present and classifying which pixels belong to which ship.

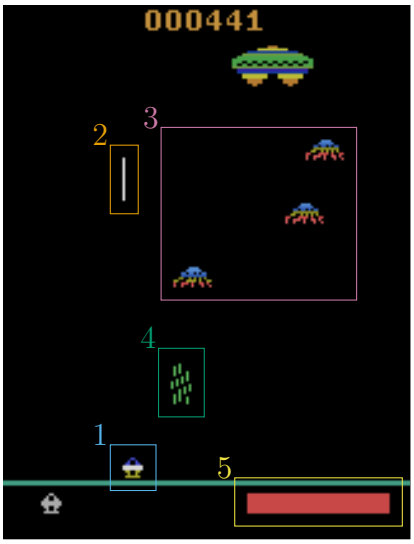

Figure 3.8: A screen from the game Assault, labeled to show the relevant entity classes listed in Table 3.1.

3.3 Density Observables

We now consider the use of the Koopman operator for entity-based systems. First, we must determine a suitable choice of dictionary. We note that, due to (3.3), it is sufficient to restrict our focus to observables that depend on the state x only through the entity-based representation $\mathscr{E}(x)$. That is, we may consider a dictionary of the form $\boldsymbol{\Psi} = \tilde{\boldsymbol{\Psi}} \circ \mathscr{E}$ for some $\tilde{\boldsymbol{\Psi}} : \mathscr{X} \to \mathbb{R}^N$. To this end, we use observables that describe a spatial "density" of entities within their individual state spaces $\mathbb{R}^{m_i}$.

First, for each entity class $i \in \{1, \ldots, n_e\}$, we choose some set of fixed points $\mathcal{B}_i = \{\mathbf{b}_{ij}\}_{k=1}^{N_i} \subseteq \mathbb{R}^{m_i}$, which we shall refer to as *basis centers*. For each basis center $\mathbf{b}_{ij}$ we define a corresponding observable ϕ_{ij} as

$$\phi_{ij}(x) := \sum_{\mathbf{e} \in \mathcal{E}_i(x)} w_{ij}(\mathbf{e}) \kappa_i(\mathbf{e}, \mathbf{b}_{ij}), \tag{3.5}$$

for a chosen weight function $w_{ij} : \mathbb{R}^{m_i} \to \mathbb{R}$ and kernel function $\kappa_i : \mathbb{R}^{m_i} \times \mathbb{R}^{m_i} \to \mathbb{R}$. We

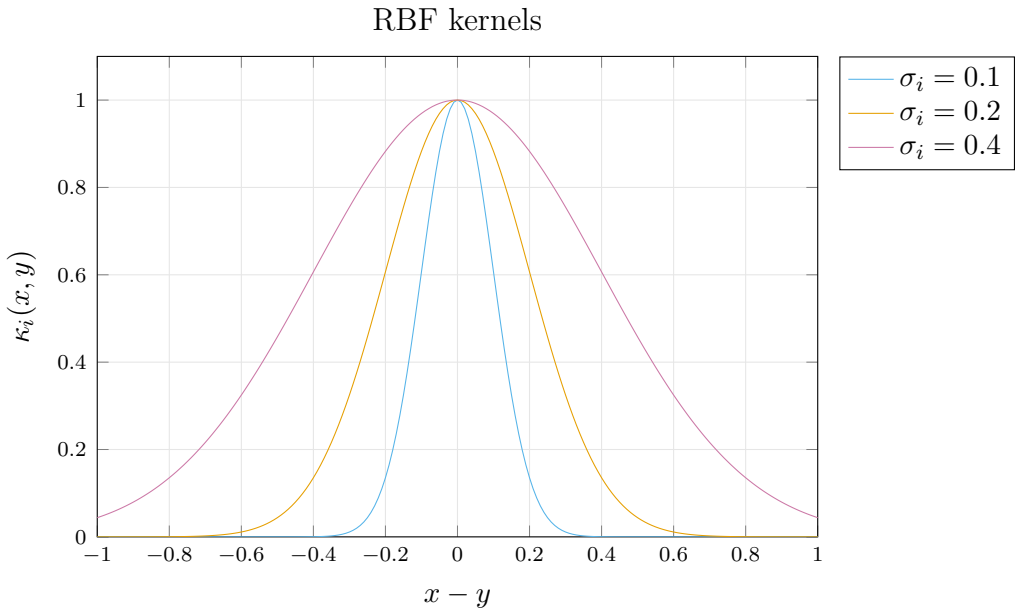

Figure 3.9: A plot of RBF kernels on $\mathbb{R}^1$, for different choices of lengthscale σ_j.

may then define a dictionary $\mathbf{\Psi} \in \mathcal{F}^N$, with $N = \sum_{j=1}^{n_e} N_i$, as

$$\mathbf{\Psi} := \begin{bmatrix} \mathbf{\Phi}_1 \\ \vdots \\ \mathbf{\Phi}_{n_e} \end{bmatrix}, \qquad \text{where} \qquad \mathbf{\Phi}_i := \begin{bmatrix} \phi_{i1} \\ \vdots \\ \phi_{iN_i} \end{bmatrix}, \quad i \in \{1, \ldots, n_e\}$$

One appropriate kernel function is the radial basis function (RBF) kernel

$$\kappa_i(\mathbf{x}, \mathbf{y}) := \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|_2^2}{2\sigma_i^2}\right), \tag{3.6}$$

where $\sigma_i > 0$ is a chosen lengthscale parameter. As shown in Figure 3.9, the RBF kernel acts as a smooth indicator of the proximity of its arguments, with $\kappa_i(\mathbf{x}, \mathbf{y}) = 1$ for $\mathbf{x} = \mathbf{y}$, and decreasing toward zero as the distance between $\mathbf{x}$ and $\mathbf{y}$ increases, at a rate determined by the parameter σ_i. Thus, each observable ϕ_{ij} describes how many entities are close to the basis center $\mathbf{b}_{ij}$. The vector $\mathbf{\Phi}_i(x)$ then captures a notion of the "density" of the entities in the corresponding set $\mathcal{E}_i(x)$, as illustrated in Figure 3.10.

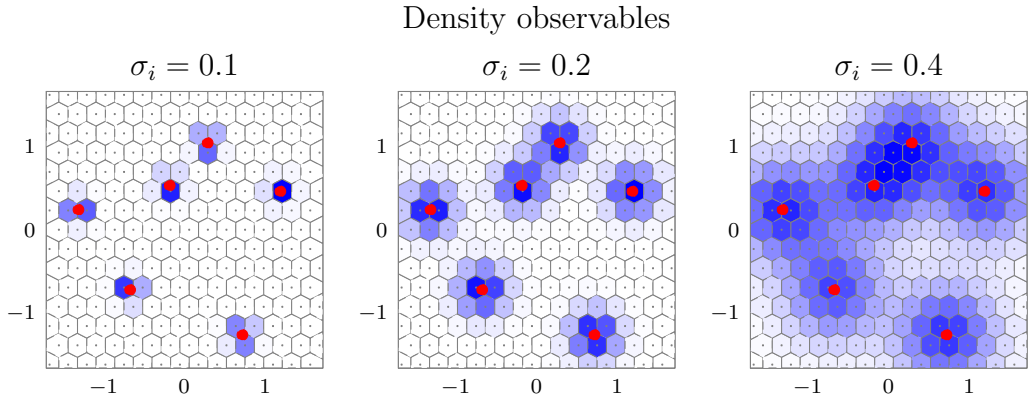

Figure 3.10: A visualization of a "density" of entities in $\mathbb{R}^2$. The red circles denote states of entities in the set $\mathcal{E}_i(x)$. A Voronoi diagram for a collection of basis centers $\mathbf{b}_{ij}$ is shown, with each cell shaded proportional to the value of the corresponding observable $\phi_{ij}(x)$. The observables use RBF kernels, with a different lengthscale σ_i used in each plot.

The weight functions w_{ij} may be omitted, which isequivalent to choosing constant weights $w_{ij}(\mathbf{e}) := 1$. However, noting that the value of the kernel $\kappa_i(\mathbf{e}, \mathbf{b}_{ij})$ is very small when the distance between $\mathbf{e}$ and $\mathbf{b}_{ij}$ is large, it can be numerically beneficial to choose weights that zero out small terms, to promote sparsity in the vector $\mathbf{\Phi}_i(x)$. For example, consider weight functions given by

$$w_{ij}(\mathbf{e}) := \begin{cases} 1, & \mathbf{b}_{ij} \in k_i\text{NN}(\mathbf{e}), \\ 0, & \text{otherwise}, \end{cases}$$

for some chosen $k_i \in \mathbb{N}$, where $k_i\text{NN}(\mathbf{e}) \subseteq \mathcal{B}_i$ denotes the set of the k_i nearest neighbors to the entity state $\mathbf{e} \in \mathcal{E}_i(x)$. This has an additional benefit of reducing the number of kernels that must be computed.

The basis centers for the observables ϕ_{ij} defined in (3.5) may be taken as any set of points $\mathcal{B}_i \subseteq \mathbb{R}^{m_i}$, e.g. via a grid in $\mathbb{R}^{m_i}$. However, this may be inefficient, especially if the entity states reside only in a small region of a high-dimensional space. We instead propose a data-driven approach that uses a set of states $\{x_k\}_{k=1}^{M_K}$ collected across one or

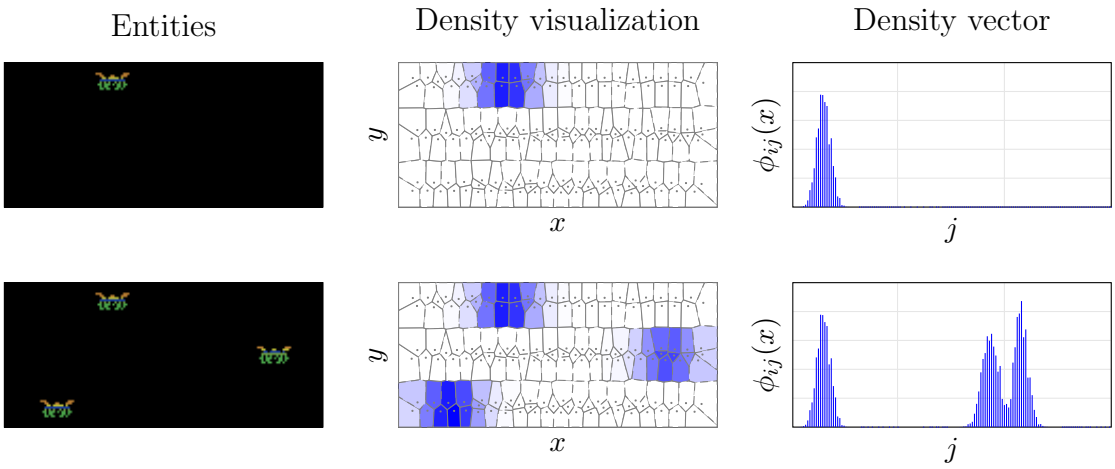

Figure 3.11: Illustration of the density of enemy ships in the game Assault. The top and bottom rows correspond to two states with different numbers of entities. On the left is a zoomed-in version of the game screen, showing the entity positions. In the middle is a visualization of the density via a Voronoi tessellation of the basis centers, with each cell shaded proportionally to the corresponding $\phi_{ij}(x)$. On the right is a plot of the same density in the form of the vector $\mathbf{\Phi}_i(x) = [\phi_{i1}(x) \ \cdots \ \phi_{iN_i}(x)]^T$.

more trajectories, akin to the set of data snapshots (2.4) used in EDMD. For each entity class $i \in \{1, \ldots, n_e\}$, we then choose the basis centers via a K-means clustering of the entity states, i.e.

$$\mathcal{B}_i = \underset{K=N_i}{K\text{-means}} \left(\bigcup_{k=1}^{M_K} \mathcal{E}_i(x_k) \right), \qquad i \in \{1, \ldots, n_e\}.$$

In this way, the basis centers will be densest in the regions of the state space that the entities are most likely to visit, enabling the use of fewer total observables to capture the most important state information.

An example of density observables for one entity class of the game Assault is illustrated in Figure 3.11. The K-means clustering results in a non-uniform placement of the basis centers, such that they are concentrated in the regions where the entities reside. Furthermore, we see that the densities are adaptable to any number of entities that may be present.

We now present an example for the MultiPong system showcasing the use of these densities observables in Koopman-based representations

Example 3.2. Consider the MultiPong system with one ball and no paddles, where states are initialized according to (3.1) with $m_x = 15$ and $m_y = 10$. Using density observables for the ball, with basis centers at each of the $4m_xm_y = 600$ reachable ball states, we can obtain an exact Koopman representation for the dynamics. We learn the system matrix A via EDMD (2.5), taking the set of snapshots (2.4) as the set of (state, next-state) pairs for an enumerated list of all reachable states. We may then obtain eigenfunctions for the system via left eigenvectors of the A matrix, as in (2.3). Such an eigenfunction with $\lambda = 1$ is illustrated in Figure 3.12.

We may also consider inexact Koopman representations by perturbing the initial states as in (3.2). For three values of δ, we learn the dynamics for the same dictionary via EDMD, with data collected over 500 episodes (50,000 total snapshots). Then, we compare the actual ball trajectory with predictions from our learned model, as shown in Figure 3.13. As the perturbation scale δ increases, the accuracy of the resulting Koopman model decreases.

3.4 Product Densities

Although the density observables we have defined provide an appropriate description of the system state, they often are not sufficient for obtaining a linear representation for the Koopman operator. In the MultiPong system for instance, the interactions between the ball and the paddle are nonlinear with respect to the separate densities. We illustrate the problem in the following example.

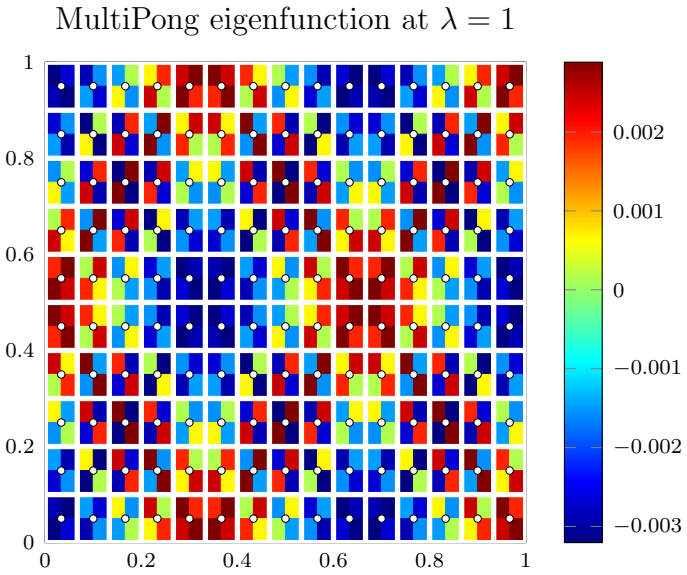

Figure 3.12: A visualization of an eigenfunction φ at $\lambda = 1$ for the MultiPong system with one ball and no paddles, with initial states (3.1) with $m_x = 15$ and $m_y = 10$. Each white dot denotes one position reachable by the ball, with the four surrounding cells shaded according to the value of $\varphi(x)$ when the ball is at that position and with the corresponding velocity in that direction. The level sets of φ correspond to distinct orbits of the ball, which are invariant sets.

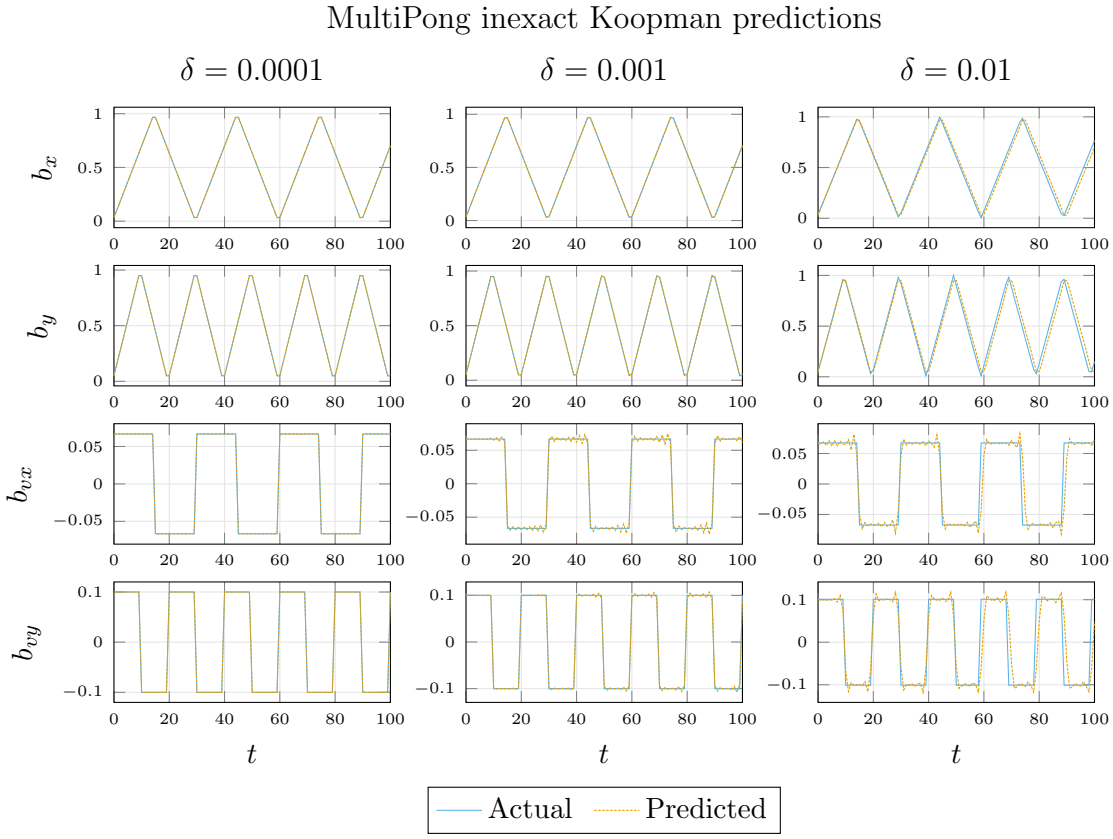

Figure 3.13: Comparison of actual and predicted trajectories using inexact Koopman representations for the MultiPong system with one ball and no paddles, using the perturbed initial velocities in (3.2) for different values of δ.

Example 3.3. Consider a minimal abstraction of the MultiPong system as described by Figure 3.14. We construct density observables for the ball and paddle, with respective basis center sets

$$\mathcal{B}_1 = \left\{ \begin{bmatrix} 1/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 1/4 \\ 3/4 \end{bmatrix}, \begin{bmatrix} 3/4 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 3/4 \\ 3/4 \end{bmatrix} \right\} \quad \text{and} \quad \mathcal{B}_2 = \left\{ \begin{bmatrix} 1/2 \\ 1/4 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 3/4 \end{bmatrix} \right\},$$

and with each using the delta kernel defined by

$$\kappa_\delta(\mathbf{x}, \mathbf{y}) := \begin{cases} 1, & \mathbf{x} = \mathbf{y}, \\ 0, & \text{otherwise.} \end{cases}$$

Each observable then acts as an indicator function for whether the corresponding entity is in the corresponding state. Now, consider the observable $\mathcal{K}\phi_{11}$. Using the state labeling in Figure 3.14, we have

$$\mathcal{K}\phi_{11}(x_i) = \begin{cases} 1, & i \in \{1, 6\}, \\ 0, & i \in \{2, 3, 4, 5, 7, 8\}. \end{cases}$$

The observable $\mathcal{K}\phi_{11}$ then indicates that either the ball is in the bottom left and the paddle is at the bottom, or the ball is in the bottom right and the paddle is at the top. The "and"s in this interpretation correspond to nonlinear functions of the separate density observables ϕ_{ij}, and so $\mathcal{K}\phi_{11}$ is not in the span of the dictionary $\mathbf{\Psi} = [\phi_{11}\ \phi_{12}\ \phi_{13}\ \phi_{14}\ \phi_{21}\ \phi_{22}]^T$.

What the previous examples show is that separate density observables on their own are not sufficient for obtaining an exact Koopman representation. The fundamental problem is that certain interactions between entities depends on their individual states in nonlinear ways. In this case, to know whether the ball will be deflected by the paddle, we require

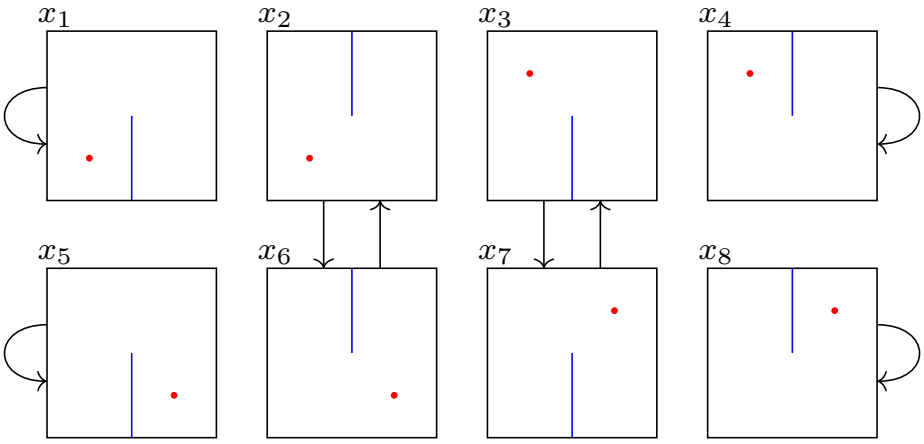

Figure 3.14: An illustration of a minimal MultiPong system, with one large paddle with two positions and one ball that alternates between the left and right unless blocked by the paddle. The eight possible states are enumerated, with arrows denoting the state transitions for the uncontrolled dynamics.

observables that depend on specific combinations of the ball and paddle positions. Such combinations may be described via products of the separate observables, as shall now define.

Consider a pair of entity classes $i, k \in \{1, \ldots, n_e\}$ with separate density observables denoted by $\mathbf{\Phi}_i = [\phi_{i1} \ \cdots \ \phi_{iN_i}]^T$ and $\mathbf{\Phi}_k = [\phi_{k1} \ \cdots \ \phi_{kN_k}]^T$. We define the *product density* $\mathbf{\Phi}_{i\otimes k} \in \mathcal{F}^{N_i N_k}$ of classes i and k by

$$\mathbf{\Phi}_{i\otimes k}(x) := \mathbf{\Phi}_i(x) \otimes \mathbf{\Phi}_k(x), \tag{3.7}$$

where $\otimes$ denotes the Kronecker product, such that

$$\mathbf{\Phi}_i(x) \otimes \mathbf{\Phi}_k(x) = \begin{bmatrix} \phi_{i1}(x)\mathbf{\Phi}_k(x) \\ \vdots \\ \phi_{iN_i}(x)\mathbf{\Phi}_k(x) \end{bmatrix} = \begin{bmatrix} \phi_{i1}(x)\phi_{k1}(x) \\ \vdots \\ \phi_{i1}(x)\phi_{kN_k}(x) \\ \vdots \\ \vdots \\ \phi_{iN_i}(x)\phi_{k1}(x) \\ \vdots \\ \phi_{iN_i}(x)\phi_{kN_k}(x) \end{bmatrix}.$$

Thus, the product density $\mathbf{\Phi}_{i\otimes k}$ consists of all observables of the form $\phi_{ij}\phi_{kl}$ for $j \in \{1, \ldots, N_i\}$ and $l \in \{1, \ldots, N_k\}$. These observables measure a joint density between the two entity class, in the sense that the value of each product observable $\phi_{ij}\phi_{kl}$ will be large only if there simultaneously exist class i entities near the basis center $\mathbf{b}_{ij}$ and class k entities near the basis center $\mathbf{b}_{kl}$.

Example 3.4. We continue the discussion of Example 3.3 regarding the system described by Figure 3.14. Consider the product of the observables ϕ_{11} (which indicates whether the ball is in the bottom left) and ϕ_{22} (which indicates whether the paddle is at the top). The product observable $\phi_{11}\phi_{22}$ is then equal to 1 only if both are true. Using the state labeling of Figure 3.14, we thus have

$$[\phi_{11}\phi_{22}](x) = \begin{cases} 1, & x = x_2, \\ 0, & \text{otherwise.} \end{cases}$$

Since $f(x) = x_2$ if and only if $x = x_6$, then we have

$$\mathcal{K}[\phi_{11}\phi_{22}](x) = [\phi_{11}\phi_{22}](f(x)) = \begin{cases} 1, & f(x) = x_2, \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1, & x = x_6, \\ 0, & \text{otherwise} \end{cases} = [\phi_{13}\phi_{22}](x),$$

and so $\mathcal{K}[\phi_{11}\phi_{22}]$ is contained in the span of $\mathbf{\Phi}_{1\otimes 2}$. A similar approach for the remaining products shows that the span is $\mathcal{K}$-invariant. Indeed, it is straightforward to identify and verify that the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

satisfies $\mathcal{K}\mathbf{\Phi}_{1\otimes 2}(x) = A\mathbf{\Phi}_{1\otimes 2}(x)$ for all states $x_1, \ldots, x_8$.

We now move from this minimal abstraction to the full MultiPong system. As with the previous examples, we find that product densities allow us to obtain an exact Koopman representation, whereas separate densities do not.

Example 3.5. Consider the MultiPong system with one ball and one paddle, where states are initialized according to (3.1) with $m_x = 5$ and $m_y = 3$. We define density observables for the ball and paddle, taking their respective sets of basis centers as the $4m_xm_y = 60$ reachable ball states and the 12 paddle states. We construct two dictionaries: $\mathbf{\Psi}_{\text{sep}} := [\mathbf{\Phi}_1^T \ \mathbf{\Phi}_2^T]^T$ containing only the separate densities, and $\mathbf{\Psi}_{\text{prod}} := \mathbf{\Phi}_{1\otimes 2}$ containing only the product densities.

For each dictionary, we learn the corresponding matrices $A(u)$ and $C(u)$ via (2.14). The set of snapshots (2.13) is generated via an enumerated list of all combinations of state and input, using the ball state as the output, i.e. $\mathbf{g}(x) = [b_x \ b_y \ b_{vx} \ b_{vy}]^T$. Figure 3.15 compares the predicted output trajectories from both Koopman models with the true system trajectories. We see that separate densities are unable to produce accurate predictions, whereas product densities enable us to predict all outputs exactly.

It should be noted that, since the Kronecker product of vectors in $\mathbb{R}^{N_i}$ and $\mathbb{R}^{N_k}$

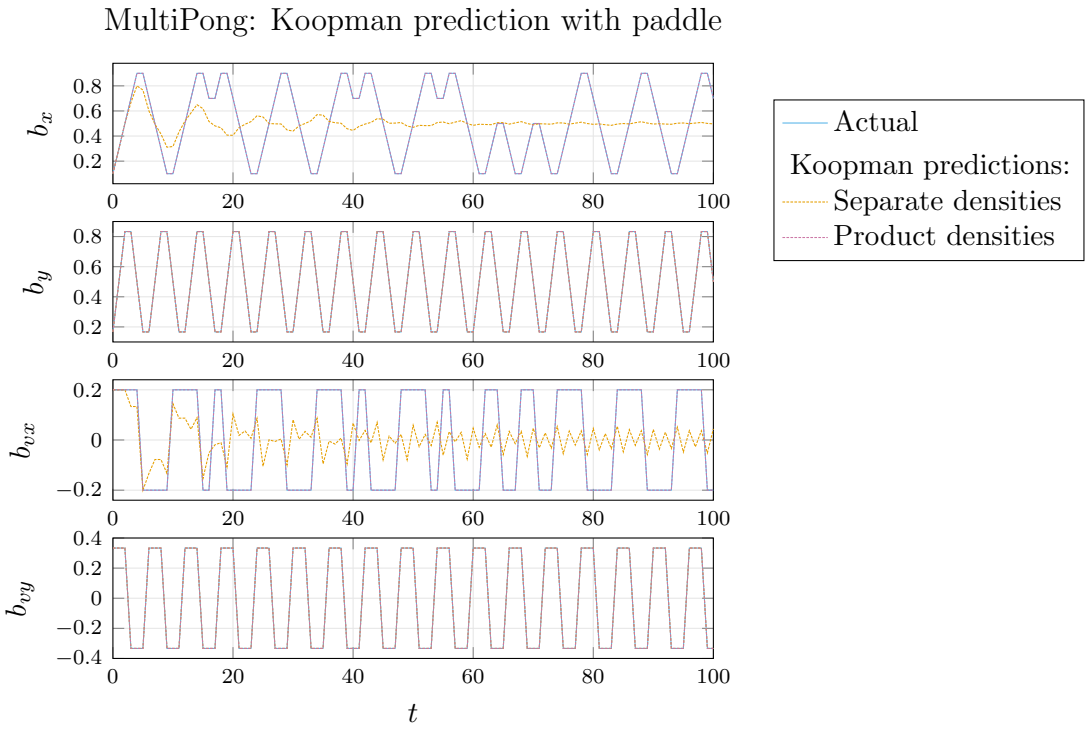

Figure 3.15: Comparison of actual trajectories for the MultiPong system with one ball and one paddle to those predicted by Koopman models with separate densities and product densities. Koopman predictions with separate densities are inaccurate beyond the first few time steps, whereas Koopman predictions with product densities match the true trajectory exactly for the entire time horizon.

produces a vector in $\mathbb{R}^{N_i N_k}$, then the product density can become very large even for moderately sized separate densities. If the size of the full Kronecker product would be impractical, one could consider using only a subset of the full Kronecker product. However, this may prove problematic, as the omitted product may be necessary for accurately predicting the included ones. An alternative is to use fewer basis centers in constructing the individual densities. Although this would reduce the precision of the individual densities, it can be worthwhile if it leads to better prediction of the product densities.

It is also interesting to examine the structure of the product observables in greater detail. Using the definition (3.5) of the individual densities, we may write

$$\begin{aligned}[\phi_{ij}\phi_{kl}](x) &= \left[\sum_{\mathbf{e}\in\mathcal{E}_i(x)} w_{ij}(\mathbf{e})\kappa_i(\mathbf{e},\mathbf{b}_{ij})\right]\left[\sum_{\mathbf{e}\in\mathcal{E}_k(x)} w_{kl}(\mathbf{e})\kappa_k(\mathbf{e},\mathbf{b}_{kl})\right] \\ &= \sum_{\mathbf{e}_i\in\mathcal{E}_i(x)}\sum_{\mathbf{e}_i\in\mathcal{E}_i(x)} w_{ij}(\mathbf{e}_i)w_{kl}(\mathbf{e}_k)\kappa_i(\mathbf{e}_i,\mathbf{b}_{ij})\kappa_k(\mathbf{e}_k,\mathbf{b}_{kl}).\end{aligned}$$

This may be simplified further depending on the specific kernel functions. For instance, the product of RBF kernels defined by (3.6) is itself an RBF kernel, since

$$\begin{aligned}\kappa_i(\mathbf{e}_i,\mathbf{b}_{ij})\kappa_k(\mathbf{e}_k,\mathbf{b}_{kl}) &= \exp\left(-\frac{\|\mathbf{e}_i-\mathbf{b}_{ij}\|_2^2}{2\sigma_i^2}\right)\exp\left(-\frac{\|\mathbf{e}_k-\mathbf{b}_{kl}\|_2^2}{2\sigma_k^2}\right) \\ &= \exp\left(-\frac{\|\mathbf{e}_i-\mathbf{b}_{ij}\|_2^2}{2\sigma_i^2}-\frac{\|\mathbf{e}_k-\mathbf{b}_{kl}\|_2^2}{2\sigma_k^2}\right) \\ &= \exp\left(-\frac{\sigma_k^2\|\mathbf{e}_i-\mathbf{b}_{ij}\|_2^2+\sigma_i^2\|\mathbf{e}_k-\mathbf{b}_{kl}\|_2^2}{2\sigma_i^2\sigma_k^2}\right) \\ &= \exp\left(-\frac{\|\sigma_k\mathbf{e}_i-\sigma_k\mathbf{b}_{ij}\|_2^2+\|\sigma_i\mathbf{e}_k-\sigma_i\mathbf{b}_{kl}\|_2^2}{2\sigma_i^2\sigma_k^2}\right) \\ &= \exp\left(-\frac{\|\mathbf{e}_{i,k}-\mathbf{b}_{ij,kl}\|_2^2}{2\sigma_i^2\sigma_k^2}\right),\end{aligned}$$

where

$$\mathbf{e}_{i,k} = \begin{bmatrix} \sigma_k \mathbf{e}_i \\ \sigma_i \mathbf{e}_k \end{bmatrix}, \qquad \mathbf{b}_{ij,kl} = \begin{bmatrix} \sigma_k \mathbf{b}_{ij} \\ \sigma_i \mathbf{b}_{kl} \end{bmatrix}.$$

With appropriate normalization, the product density in (3.7) may then be expressed equivalently in the form (3.5) on a set of combined entities

$$\left\{ \begin{bmatrix} \mathbf{e}_i \\ \mathbf{e}_k \end{bmatrix} \in \mathbb{R}^{m_i + m_k} \ : \ \mathbf{e}_i \in \mathcal{E}_i(x), \mathbf{e}_k \in \mathcal{E}_k(x) \right\}.$$

In fact, this set constitutes a valid entity class as mathematically defined in Definition 3.1, and so our description of product densities is in some sense not necessary. However, the product density interpretation allows us to retain separate, intuitively defined entity classes.

Chapter 4

Koopman-Based Controller Design

This chapter discusses the application of our Koopman-based representations for control. For a nonlinear optimal control problem, we define an equivalent linear problem via Koopman representations, from which we derive a compact, piecewise linear solution. Even with this reformulation, finding a global optimum has exponential complexity in the problem horizon, which we circumvent with a heuristic algorithm for finding a computationally tractable local solution. Lastly, we present extensions to the problem to incorporate state constraints and robustness.

4.1 Optimal Control

Consider a dynamical system with state $x \in \mathcal{X}$ and input $u \in \mathcal{U}$ described by

$$x^+ = f(x, u). \tag{4.1}$$

Given an initial state x_0, we wish to solve the associated optimal control problem

$$\min_{u_0,\ldots,u_{h-1}\in\mathcal{U}} \quad \gamma_h(x_h) + \sum_{t=0}^{h-1} \gamma_t(x_t, u_t), \tag{4.2a}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t), \quad t \in \{0, \ldots, h-1\}, \tag{4.2b}$$

where the function $\gamma_h : \mathcal{X} \to \mathbb{R}$ denotes a terminal cost, and the functions $\gamma_t : \mathcal{X} \times \mathcal{U} \to \mathbb{R},\ t \in \{0, \ldots, h-1\}$ denote cumulative stage costs.

The associated optimal cost-to-go from time t is described by the function $J_t^{\mathrm{NL}} : \mathcal{X} \to \mathbb{R}$ defined by

$$J_t^{\mathrm{NL}}(x_t) := \min_{u_t,\ldots,u_{h-1}\in\mathcal{U}} \quad \gamma_h(x_h) + \sum_{\tau=t}^{h-1} \gamma_t(x_\tau, u_\tau) \tag{4.3a}$$

$$\text{s.t.} \quad x_{\tau+1} = f(x_\tau, u_\tau), \quad \tau \in \{t, \ldots, h-1\}. \tag{4.3b}$$

Based on principles of dynamic programming [7], this cost-to-go satisfies the recursive relation

$$\begin{aligned} J_h^{\mathrm{NL}}(x_h) &= \gamma_h(x_h), \\ J_t^{\mathrm{NL}}(x_t) &= \min_{u_t\in\mathcal{U}} \left\{\gamma_t(x_t, u_t) + J_{t+1}^{\mathrm{NL}}(f(x_t, u_t))\right\}, \qquad \forall x_t \in \mathcal{X},\ \forall t \in \{0, \ldots, h-1\}. \end{aligned} \tag{4.4}$$

If the state set $\mathcal{X}$ has a finite, computationally tractable size, one can use this relation to explicitly compute the value of $J_t^{\mathrm{NL}}(x_t)$ for every $x_t \in \mathcal{X}$ and $t \in \{0, \ldots, h-1\}$. Otherwise, we require some special structure of the dynamics and cost functions to obtain an explicit formula, or a convex problem that can be efficiently solved by numerical methods. In general, however, the nonlinearities give rise to a nonconvex optimization problem for which no efficient solution exists.

The Koopman operator provides us with a way to transform the nonlinear problem (4.2) into an equivalent linear one. This is possible if we can obtain an exact Koopman representation (in the sense of Definition 2.7) for the system

$$x^+ = f(x,u), \tag{4.5a}$$

$$\mathbf{y} = \mathbf{g}(x,u) := \begin{bmatrix}\gamma_0(x,u) & \cdots & \gamma_{h-1}(x,u) & \gamma_h(x)\end{bmatrix}^T. \tag{4.5b}$$

Proposition 4.1. *Suppose* $(\mathbf{\Psi}, A, C)$ *is an exact Koopman representation for* (4.5) *where*

$$C(u) = \begin{bmatrix}\mathbf{c}_0(u)^T \\ \vdots \\ \mathbf{c}_{h-1}(u)^T \\ \mathbf{c}_h^T\end{bmatrix}, \qquad \forall u \in \mathcal{U},$$

for mappings $\mathbf{c}_t : \mathcal{U} \to \mathbb{R}^N,\ t \in \{0,\dots,h-1\}$ *and a vector* $\mathbf{c}_h \in \mathbb{R}^N$*. Define the functions* $J_t^{\mathrm{K}} : \mathbb{R}^N \to \mathbb{R},\ t \in \{0,\dots,h\}$ *by*

$$J_t^{\mathrm{K}}(\boldsymbol{\xi}_t) := \min_{u_t,\dots,u_{h-1}\in\mathcal{U}} \quad \mathbf{c}_h^T\boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T\boldsymbol{\xi}_\tau \tag{4.6a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_{\tau+1} = A(u_\tau)\boldsymbol{\xi}_\tau, \quad \tau \in \{t,\dots,h-1\}. \tag{4.6b}$$

Then

$$J_t^{\mathrm{NL}}(x_t) = J_t^{\mathrm{K}}(\mathbf{\Psi}(x_t)), \qquad \forall x_t \in \mathcal{X},\ t \in \{0,\dots,h\}.$$

Proof: Fix an initial state $x_t \in \mathcal{X}$ and a sequence of inputs $u_t,\dots,u_{h-1} \in \mathcal{U}$, and let $\boldsymbol{\xi}_t = \mathbf{\Psi}(x_t)$. Then, consider the states $x_\tau \in \mathcal{X}$ and lifted states $\boldsymbol{\xi}_\tau \in \mathbb{R}^N$ obtained by recursive application of (4.3b) and (4.6b) respectively.

Suppose, for some $\tau \in \{t,\dots,h-1\}$, that $\boldsymbol{\xi}_\tau = \mathbf{\Psi}(x_\tau)$. Then, since $(\mathbf{\Psi}, A, C)$ is an

exact Koopman representation, we have

$$\boldsymbol{\xi}_{\tau+1} = A(u_\tau)\boldsymbol{\xi}_\tau = A(u_\tau)\boldsymbol{\Psi}(x_\tau) = \mathcal{K}(u_\tau)\boldsymbol{\Psi}(x_\tau) = \boldsymbol{\Psi}(f(x_\tau, u_\tau)) = \boldsymbol{\Psi}(x_{\tau+1}).$$

Thus, by induction we have $\boldsymbol{\xi}_\tau = \boldsymbol{\Psi}(x_\tau)$ for all $\tau \in \{t, \dots, h\}$.

Now, since $(\boldsymbol{\Psi}, A, C)$ is an exact Koopman representation, we have

$$\begin{bmatrix} \gamma_0(x,u) \\ \vdots \\ \gamma_{h-1}(x,u) \\ \gamma_h(x) \end{bmatrix} = \begin{bmatrix} \mathbf{c}_0(u)^T \\ \vdots \\ \mathbf{c}_{h-1}(u)^T \\ \mathbf{c}_h^T \end{bmatrix} \boldsymbol{\Psi}(x), \qquad \forall x \in \mathcal{X},\ \forall u \in \mathcal{U},$$

and so

$$\gamma_h(x_h) + \sum_{\tau=t}^{h-1} \gamma_\tau(x_\tau, u_\tau) = \mathbf{c}_h^T \boldsymbol{\Psi}(x_h) + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \boldsymbol{\Psi}(x_\tau) = \mathbf{c}_h^T \boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \boldsymbol{\xi}_\tau.$$

Since this holds for any $x_t \in \mathcal{X}$ and $u_t, \dots, u_{h-1} \in \mathcal{U}$, we conclude that $J_t^{\mathrm{NL}}(x_t) = J_t^{\mathrm{K}}(\boldsymbol{\Psi}(x_t))$ as desired. ■

We have shown that, if we can obtain an exact Koopman representation for the system (4.5), then we can equivalently express the nonlinear cost-to-go function (4.3) for the problem (4.2) by the linear cost-to-go function (4.6). The benefit in doing this is that this linearity enables a compact, computationally efficient solution to the original problem. To express these results, it is helpful to first introduce more compact notation for discussing finite sequences of inputs.

Given $s, t \in \mathbb{N}$ with $s \leq t$, we use the notation $\boldsymbol{\mu}_s^t$ as an abbreviation of the input sequence $u_s u_{s+1} \cdots u_{t-1} \in \mathcal{U}^{t-s}$. Note that u_s is included at the start but that u_t is excluded from the end, so that $\boldsymbol{\mu}_s^t$ denotes the input sequence that transitions the system (4.1) from the state x_s at time s to the state x_t at time t. For the case $s = t$, this

represents the empty sequence, which we denote by ϵ.

We let $\mathcal{U}^*$ denote the set of all finite input sequences (including the empty sequence), i.e. $\mathcal{U}^* := \bigcup_{k=0}^{\infty} \mathcal{U}^k$ with the convention that $\mathcal{U}^0 = \{\epsilon\}$. Given a matrix-valued function $A : \mathcal{U} \to \mathbb{R}^{N\times N}$, we denote its domain extension from inputs $u \in \mathcal{U}$ to input sequences $\boldsymbol{\mu} \in \mathcal{U}^*$ by the function $\bar{A} : \mathcal{U}^* \to \mathbb{R}^{N\times N}$, defined by

$$\begin{aligned} \bar{A}(\epsilon) &:= I_N, \\ \bar{A}(u\boldsymbol{\mu}) &:= \bar{A}(\boldsymbol{\mu})A(u), \qquad \forall u \in \mathcal{U},\ \forall \boldsymbol{\mu} \in \mathcal{U}^*. \end{aligned}$$

Note the reversed order between the sequence concatenation and the matrix products. In this way, we have

$$\bar{A}(\boldsymbol{\mu}_s^t) = \bar{A}(u_s u_{s+1} \cdots u_{t-1}) = A(u_{t-1}) \cdots A(u_{s+1})A(u_s),$$

so that $\boldsymbol{\Psi}(x_t) = \bar{A}(\boldsymbol{\mu}_s^t)\boldsymbol{\Psi}(x_s)$ for an exact Koopman representation $(\boldsymbol{\Psi}, A, C)$.

Using this notation, we may state the following result.

Theorem 4.2. *The cost-to-go function J_t^{K} as defined by* (4.6) *is piecewise linear and given by*

$$J_t^{\mathrm{K}}(\boldsymbol{\xi}_t) = \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \mathbf{d}(\boldsymbol{\mu}_t^h)^T \boldsymbol{\xi}_t, \tag{4.7}$$

where $\mathbf{d} : \mathcal{U}^* \to \mathbb{R}^N$ is defined by

$$\mathbf{d}(\boldsymbol{\mu}_t^h) := \bar{A}(\boldsymbol{\mu}_t^h)^T \mathbf{c}_h + \sum_{\tau=t}^{h-1} \bar{A}(\boldsymbol{\mu}_t^\tau)^T \mathbf{c}_\tau(u_\tau). \tag{4.8}$$

Proof: For a fixed input sequence $\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}$, recursive application of (4.6b) gives

$$\boldsymbol{\xi}_\tau = \bar{A}(\boldsymbol{\mu}_t^\tau)\boldsymbol{\xi}_t, \qquad \forall \tau \in \{t, \ldots, h\}\,.$$

Thus, the objective (4.6a) to be minimized is given by

$$\begin{aligned}\mathbf{c}_h^T\boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1}\mathbf{c}_\tau(u_\tau)^T\boldsymbol{\xi}_\tau &= \mathbf{c}_h^T\bar{A}(\boldsymbol{\mu}_t^h)\boldsymbol{\xi}_t + \sum_{\tau=t}^{h-1}\mathbf{c}_\tau(u_\tau)^T\bar{A}(\boldsymbol{\mu}_t^\tau)\boldsymbol{\xi}_t \\ &= \left(\bar{A}(\boldsymbol{\mu}_t^h)^T\mathbf{c}_h + \sum_{\tau=t}^{h-1}\bar{A}(\boldsymbol{\mu}_t^\tau)^T\mathbf{c}_\tau(u_\tau)\right)^T\boldsymbol{\xi}_t \\ &= \mathbf{d}(\boldsymbol{\mu}_t^h)^T\boldsymbol{\xi}_t.\end{aligned}$$

■

A key advantage of this form is that the $\mathbf{d}(\boldsymbol{\mu})$ vectors are fully independent of the state, and thus they may be precomputed offline. A runtime controller then only needs to perform the lifting $\boldsymbol{\Psi}(x_0)$ for the initial state, and then compute the inner product $\mathbf{d}(\boldsymbol{\mu})^T\boldsymbol{\Psi}(x_0)$ for each input sequence $\boldsymbol{\mu} \in \mathcal{U}^h$. In this way, we evaluate the cost without explicitly computing the lifted state trajectory, which would require a sequence of costly matrix-vector products $\boldsymbol{\xi}_{t+1} = A(u_t)\boldsymbol{\xi}_t$.

We further note that the piecewise linear structure of the cost-to-go J_t^{K} is non-restrictive, since the dictionary $\boldsymbol{\Psi}$ will typically include nonlinear functions of the state. For example, although the pointwise minimum of linear functions is always concave, this applies only when viewed as a function of the lifted state $\boldsymbol{\Psi}(x)$. However, when instead viewed as a function of the original state x, then a nonlinear lifting yields a pointwise minimum of nonlinear functions, which may be non-concave in general.

Example 4.3. Consider a system with state $\mathbf{x} = [x_1\ x_2]^T \in \mathbb{R}^2$ and input $u \in \{1, 2\}$, with dynamics given by

$$\mathbf{x}^+ = f(\mathbf{x}, u) = \begin{cases} \begin{bmatrix} 1.2x_1 \\ -0.8x_2 + 0.6x_1^2 \end{bmatrix}, & u = 1, \\ \begin{bmatrix} -0.4x_1 \\ 0.8x_2 - 0.2x_1^2 \end{bmatrix}, & u = 2, \end{cases}$$

and an associated optimal control problem (4.2) with horizon $h = 10$ and cost functions

$$\gamma_t(\mathbf{x}, u) = \begin{cases} tx_1 + x_2, & u = 1, \\ x_1 + tx_2, & u = 2, \end{cases} \qquad \gamma_h(\mathbf{x}) = 10x_1^2.$$

The dictionary $\boldsymbol{\Psi}(\mathbf{x}) = [x_1 \; x_2 \; x_1^2]^T$ admits an exact Koopman representation, as we have $\mathcal{K}(u)\boldsymbol{\Psi}(\mathbf{x}) = A(u)\boldsymbol{\Psi}(\mathbf{x})$ where

$$A(1) = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & -0.8 & 0.6 \\ 0 & 0 & 1.44 \end{bmatrix}, \qquad A(2) = \begin{bmatrix} -0.4 & 0 & 0 \\ 0 & 0.8 & -0.2 \\ 0 & 0 & 0.16 \end{bmatrix},$$

and that $\gamma_t(\mathbf{x}, u) = \mathbf{c}_t(u)^T\boldsymbol{\Psi}(\mathbf{x})$ and $\gamma_h(\mathbf{x}) = \mathbf{c}_h^T\boldsymbol{\Psi}(\mathbf{x})$ where

$$\mathbf{c}_t(1) = \begin{bmatrix} t \\ 1 \\ 0 \end{bmatrix}, \qquad \mathbf{c}_t(2) = \begin{bmatrix} 1 \\ t \\ 0 \end{bmatrix}, \qquad \mathbf{c}_h = \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix}.$$

Thus, the original nonlinear problem is equivalent to the linear problem (4.6), which may be solved via the compact form (4.7) provided by Theorem 4.2. A visualization of the solution structure is provided in Figure 4.1.

4.2 Dynamic Pruning Algorithm

For small horizon lengths h, it is feasible to compute the vectors $\mathbf{d}(\boldsymbol{\mu})$ that define the cost-to-go (4.7) for every input sequence $\boldsymbol{\mu} \in \mathcal{U}^h$. However, this full enumeration quickly becomes intractable for longer horizons, since the total number of input sequences grows exponentially in h. If one insists on finding the global optimum, then this complexity cannot be avoided, as the problem is known to be NP-hard in general [66]. Instead, we implement a heuristic algorithm that constructs a tractable subset of sequences by dy-

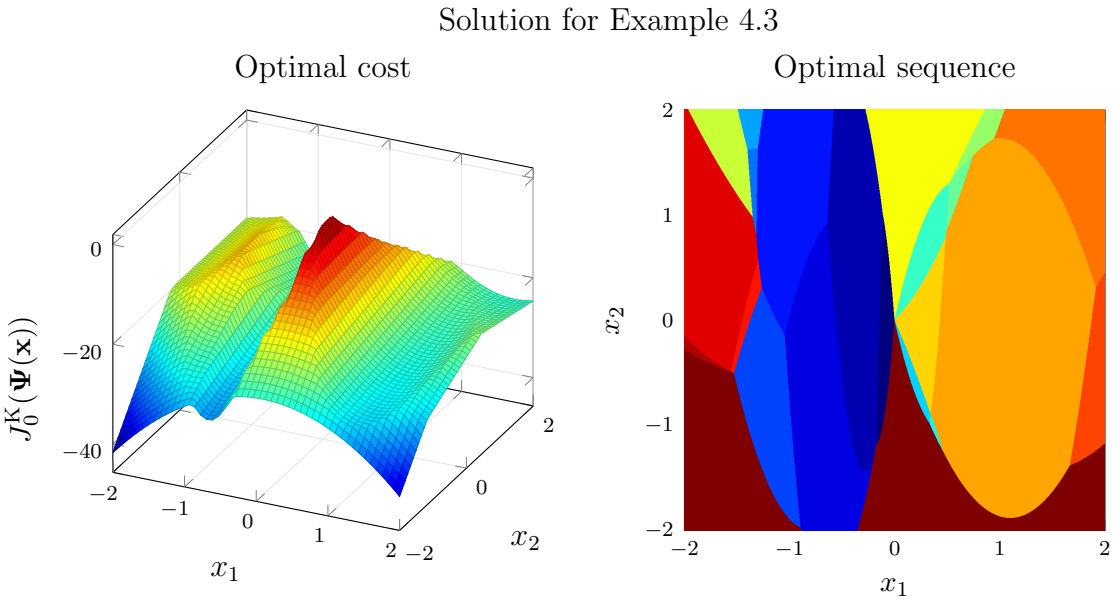

Figure 4.1: An illustration of the solution to the optimal control problem posed in Example 4.3. Left: The optimal cost as a function of the initial state $\mathbf{x}$. The parabolic surfaces in the original state space correspond to hyperplanes in the lifted state space defined by the vectors $\mathbf{d}(\boldsymbol{\mu})$. Right: A partition of the state space according to the optimal input sequence, colored blue to red in the lexicographical ordering of the sequences. Out of $2^{10} = 1{,}024$ total sequences, only 22 are optimal from any state in the domain shown.

namically pruning the search space via a combination "exploitation" and "exploration".

4.2.1 Central Idea

Although the cost-to-go function (4.6) and the equivalent expression (4.7) are defined in terms of exponentially many $\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}$, it may be the case that relatively few are ever optimal, as we saw in Example 4.3. In principle, there should then exist some much smaller subset of sequences for which the minimal cost remains optimal or near-optimal. We propose here a heuristic algorithm for constructing such subsets $\mathtt{U}_t \subseteq \mathcal{U}^{h-t}$.

We note that the vectors $\mathbf{d}(\boldsymbol{\mu}_t^h)$ defined by (4.8) follow the recursive relation

$$\mathbf{d}(\boldsymbol{\mu}_t^h) = \mathbf{c}_t(u_t) + A(u_t)^T \mathbf{d}(\boldsymbol{\mu}_{t+1}^h), \qquad t \in \{0, \dots, h-1\} \tag{4.9}$$

similar to the Bellman equation (4.4) for the cost-to-go. With the initial condition $\mathbf{d}(\epsilon) = \mathbf{c}_h$, we may then recursively construct the vectors backward in time via (4.9). It is reasonable then to construct the subsets $\mathtt{U}_t \subseteq \mathcal{U}^{h-t}$ in parallel such that

$$\mathtt{U}_t \subseteq \mathcal{U} \cdot \mathtt{U}_{t+1} := \{u\boldsymbol{\mu} : u \in \mathcal{U}, \boldsymbol{\mu} \in \mathtt{U}_{t+1}\}.$$

In this way, the computation of the vectors $\{\mathbf{d}(\boldsymbol{\mu}_t^h) : \boldsymbol{\mu}_t^h \in \mathtt{U}_t\}$ via (4.9) relies only on those vectors in the set $\{\mathbf{d}(\boldsymbol{\mu}_{t+1}^h) : \boldsymbol{\mu}_{t+1}^h \in \mathtt{U}_{t+1}\}$.

For times t near the end of the horizon, the full set of possible sequences $\mathcal{U}^{h-t}$ is relatively small, and so it is feasible to consider the full subset, i.e. to let $\mathtt{U}_t = \mathcal{U} \cdot \mathtt{U}_{t+1}$. As we move backward in time, the total number of sequences grows exponentially, and we must instead take $\mathtt{U}_t$ as some "pruned" subset of $\mathcal{U} \cdot \mathtt{U}_{t+1}$. We choose this pruned subset via a heuristic algorithm that combines the concepts of "exploitation" and "exploration" as commonly considered in reinforcement learning. For exploitation, we use the already

computed vectors $\mathbf{d}(\boldsymbol{\mu}^h_{t+1})$ to dynamically evaluate the costs of the associated sequences $\boldsymbol{\mu}^h_t = u_t \boldsymbol{\mu}^h_{t+1}$ from some number of test states, and we save those that perform well. For exploration, we randomly select some number of sequences from $\mathcal{U} \cdot \mathtt{U}_{t+1}$ to be saved in the pruned set $\mathtt{U}_t$.

4.2.2 The Algorithm

The main pruning algorithm is that of Algorithm 1, which contains the high level logic and calls Algorithm 2 as a subroutine to perform pruning as needed. The algorithm depends on a number of chosen parameters as follows: a fixed capacity $L \in \mathbb{N}$ that denotes the maximum number of sequences to be saved in the set $\mathtt{U}_t$ at each iteration t; a set of initial states $\mathtt{X}_0 = \{x_1, \ldots, x_{M_x}\} \subseteq \mathcal{X}$ for dynamic cost evaluation; and parameters $M_b, M_r \in \mathbb{N}$ that denote how many sequences should be saved in the exploitation and exploration phases respectively. To ensure that the size of the sets $\mathtt{U}_t$ remain within the capacity L, these parameters must be chosen such that

$$hM_bM_x + M_r \leq L. \tag{4.10}$$

Algorithm 1 recursively constructs the subsets $\mathtt{U}_t$ backward from the end of the horizon. The PruneStep function defined by Algorithm 2 is called to perform pruning if the full set $\mathcal{U} \cdot \mathtt{U}_{t+1}$ would be larger than the capacity. At each iteration, it also computes (via (4.9)) the corresponding vectors $\mathbf{d}_{\boldsymbol{\mu}}$ for each input sequence $\boldsymbol{\mu}$ saved in the pruned set.

The PruneStep function defined in Algorithm 2 takes a set of input sequences $\mathtt{U}_{t+1}$, and a corresponding data set $\mathtt{D}_{t+1} = \{\mathbf{d}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathtt{U}_{t+1}\}$, and returns a subset of extended sequences from $\mathcal{U} \cdot \mathtt{U}_{t+1}$ constructed in two phases. The first phase, illustrated in Figure 4.2, is an "exploitation" phase, in which we use the vectors $\mathbf{d}_{\boldsymbol{\mu}} \in \mathtt{D}_{t+1}$ to evaluate

Algorithm 1 Dynamic Pruning

$\mathtt{U}_h \leftarrow \{\epsilon\}$
$\mathbf{d}_\epsilon \leftarrow \mathbf{c}_h$
for $t \in \{h-1, \dots, 0\}$ **do**
    if $n_u|\mathtt{U}_{t+1}| \leq L$ **then**
        $\mathtt{U}_t \leftarrow \mathcal{U} \cdot \mathtt{U}_{t+1}$
    else
        $\mathtt{U}_t \leftarrow \textsc{PruneStep}(\mathtt{U}_{t+1}, \mathtt{D}_{t+1})$
    for $u\boldsymbol{\mu} \in \mathtt{U}_t$ **do**
        $\mathbf{d}_{u\boldsymbol{\mu}} \leftarrow A(u)^T \mathbf{d}_{\boldsymbol{\mu}} + \mathbf{c}_t(u)$
    $\mathtt{D}_t \leftarrow \{\mathbf{d}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathtt{U}_t\}$
return $\mathtt{U}_0, \mathtt{D}_0$

the cost associated with each sequence $u\boldsymbol{\mu} \in \mathcal{U} \cdot \mathtt{U}_{t+1}$ for a number of test states. These test states are dynamically generated in a receding horizon manner from each state in the set $\mathtt{X}_0 = \{x_1, \dots, x_{M_x}\} \subseteq \mathcal{X}$. That is, for each $x_j \in \mathtt{X}_0$, we let $\boldsymbol{\xi}_{j,0} = \boldsymbol{\Psi}(x_j)$, from which generate a subsequent state as $\boldsymbol{\xi}_{j,1} = A(u^*)\boldsymbol{\xi}_{j,0}$ where $u^* = \arg\min_{u \in \mathcal{U}} \{\mathbf{d}_{\boldsymbol{\mu}}{}^T A(u)\boldsymbol{\xi}_{j,0} + \mathbf{c}_t(u)^T \boldsymbol{\xi}_{j,0}\}$. We obtain a set of lifted states $\boldsymbol{\xi}_{j,k}$ by repeating this procedure, and for each state we save the M_b sequences that achieved the lowest costs. The second "exploration" phase of the algorithm simply chooses M_r additional sequences at random to be included in the pruned set.

When Algorithm 1 terminates, it returns a subset $\mathtt{U}_0 \subseteq \mathcal{U}^h$ and the corresponding set of vectors $\mathtt{D}_0 = \{\mathbf{d}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathtt{U}_0\}$. Using these, we define

$$\widehat{J}_0^{\mathrm{K}}(\boldsymbol{\xi}_0) = \min_{\boldsymbol{\mu} \in \mathtt{U}_0} \mathbf{d}_{\boldsymbol{\mu}}{}^T \boldsymbol{\xi}_0, \tag{4.11}$$

which serves as a computationally tractable approximation of (4.7).

Example 4.4. Consider the MultiPong system with one ball and paddle, initialized via (3.1) with $m_x = 5$ and $m_y = 3$. We examine the approximation error of the pruned

Algorithm 2 PRUNESTEP($\mathtt{U}_{t+1}, \mathtt{D}_{t+1}$)

Require: $\mathtt{U}_{t+1} \subseteq \mathcal{U}^{h-(t+1)}$, $\mathtt{D}_{t+1} = \{\mathbf{d}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathtt{U}_{t+1}\}$

for $x_j \in \mathtt{X}_0$ **do**
  $\boldsymbol{\xi}_{j,0} \leftarrow \boldsymbol{\Psi}(x_j)$
  for $k \in \{0, \dots, h-1\}$ **do**
    for $u \in \mathcal{U}$ **do**
      $\boldsymbol{\xi}^u_{j,k} \leftarrow A(u)\boldsymbol{\xi}_{j,k}$
      $c^u_{j,k} \leftarrow \mathbf{c}_t(u)^T \boldsymbol{\xi}_{j,k}$
      for $\boldsymbol{\mu} \in \mathtt{U}_{t+1}$ **do**
        $J_{u\boldsymbol{\mu}} \leftarrow \mathbf{d}_{\boldsymbol{\mu}}{}^T \boldsymbol{\xi}^u_{j,k} + c^u_{j,k}$
    $\mathtt{U}^{j,k}_{\text{best}} \leftarrow$ Set of sequences $u\boldsymbol{\mu} \in \mathcal{U} \cdot \mathtt{U}_{t+1}$ that achieve the M_b lowest costs $J_{u\boldsymbol{\mu}}$
    $u^* \leftarrow \arg\min_{u \in \mathcal{U}} \min_{\boldsymbol{\mu} \in \mathtt{U}_{t+1}} J_{u\boldsymbol{\mu}}$
    $\boldsymbol{\xi}_{j,k+1} \leftarrow \boldsymbol{\xi}^{u^*}_{j,k}$
$\mathtt{U}_{\text{best}} \leftarrow \bigcup_{j=1}^{M_x} \bigcup_{k=0}^{h-1} \mathtt{U}^{j,k}_{\text{best}}$
$\mathtt{U}_{\text{rand}} \leftarrow \{M_r \text{ random samples from } \mathcal{U} \cdot \mathtt{U}_{t+1}\}$
return $\mathtt{U}_{\text{best}} \cup \mathtt{U}_{\text{rand}}$

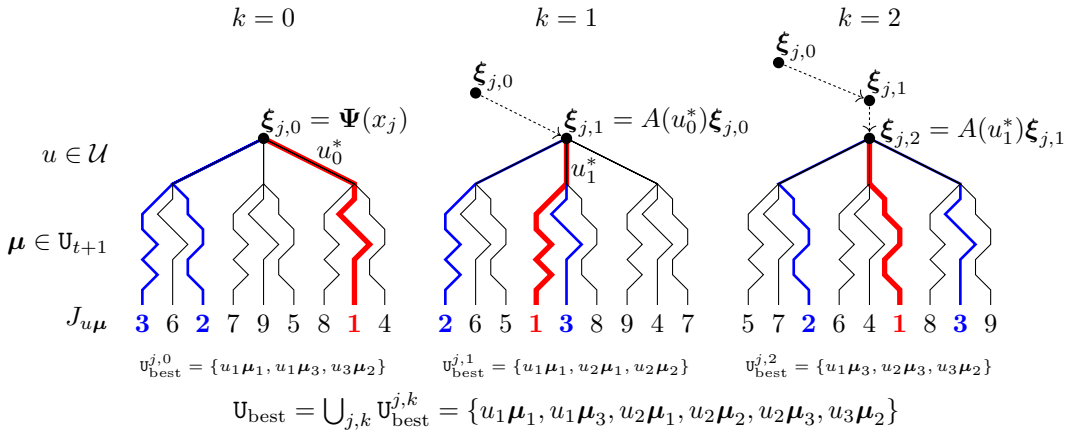

Figure 4.2: An illustrated example of the first three iterations of the "exploitation" phase of Algorithm 2, for one initial state x_j and with $M_b = 3$. For each k, we compute the cost associated to each sequence $u\boldsymbol{\mu} \in \mathcal{U} \cdot \mathtt{U}_{t+1}$ from the state $\boldsymbol{\xi}_{j,k}$ as $J_{u\boldsymbol{\mu}} = \mathbf{d}_{\boldsymbol{\mu}}{}^T A(u)\boldsymbol{\xi}_{j,k} + \mathbf{c}_t(u)^T\boldsymbol{\xi}_{j,k} = \mathbf{d}_{u\boldsymbol{\mu}}{}^T\boldsymbol{\xi}_{j,k}$. The first input of the best sequence (shown in red) is used to generate the test state $\boldsymbol{\xi}_{j,k+1}$ for the next iteration. At the same time, the M_b best sequences (the best sequence, together with the next best sequences shown in blue) are saved as the set $\mathtt{U}^{j,k}_{\text{best}}$, and their union is taken as the set $\mathtt{U}_{\text{best}}$.

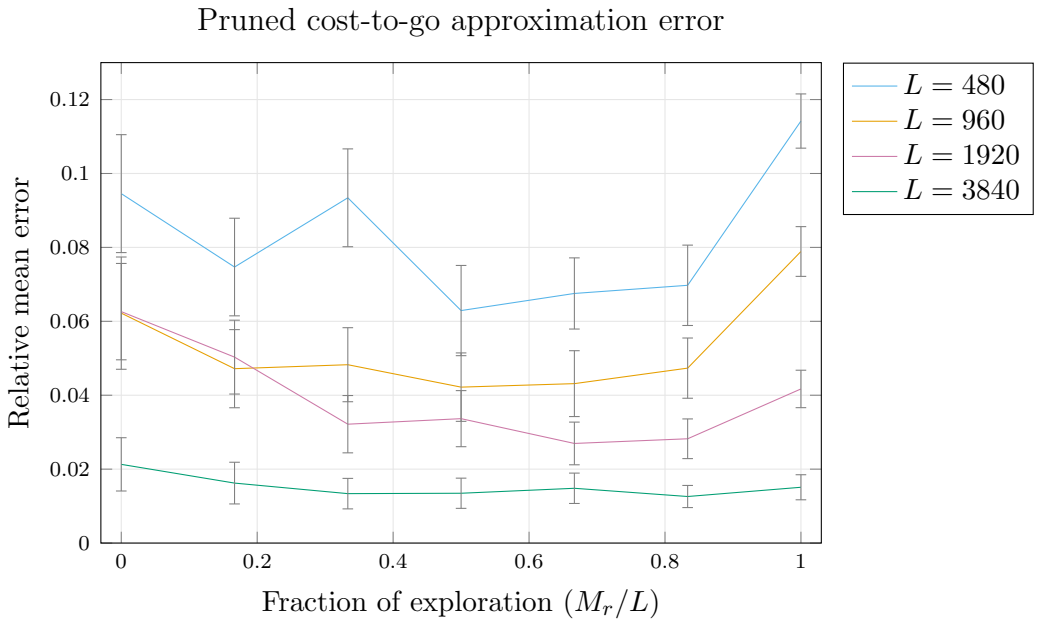

Figure 4.3: Relative approximation error between the pruned cost-to-go function (4.11) and the unpruned cost-to-go function (4.7), computed as $\|\widehat{J}_0^{\mathrm{K}} - J_0^{\mathrm{K}}\|_2 / \|J_0^{\mathrm{K}}\|_2$.

cost-to-go (4.11) for an optimal control problem with horizon $h = 8$. This horizon is sufficiently short that we are able to compute the unpruned cost-to-go J_0^{K} via (4.7). Using Algorithm 1, we also construct pruned versions of the cost-to-go $\widehat{J}_0^{\mathrm{K}}$ defined by (4.11), for several different values of the parameters L, M_b and M_r. We then compute the relative mean error of each pruned function, shown in Figure 4.3. Larger capacities L achieve the most accurate approximations, but even smaller capacities achieve low errors. We also see that a balance between exploitation and exploration consistently yield better accuracy than pure exploitation or pure exploration.

4.2.3 Computational Complexity

We now analyze the complexity of the dynamic pruning algorithm, working outward from the innermost loop of Algorithm 2. The computation of $\mathbf{d}_{\boldsymbol{\mu}}{}^T \boldsymbol{\xi}_{j,k}^u$ in step 8 requires N operations, and is performed for L sequences $\boldsymbol{\mu} \in \mathtt{U}_{t+1}$. The computation of $A(u)\boldsymbol{\xi}_{j,k}$

Table 4.1: The computational complexity for the pruned and unpruned cost-to-go functions.

	Full cost-to-go	Pruned cost-to-go
Offline construction	$O({n_u}^h N^2)$	$O(h^2 n_u M_x[N^2 + LN] + hLN^2)$
Runtime evaluation	$O({n_u}^h N)$	$O(LN)$

in step 5 requires an additional N^2 operations. These computations are repeated for the n_u inputs $u \in \mathcal{U}$, for h values of k, and for M_x states $x_j \in \mathtt{X}_0$. Thus, the complexity of Algorithm 2 is $O(hn_u M_x[N^2 + LN])$.

We may now consider Algorithm 1. The computation of $A(u)^T \mathbf{d}_{\boldsymbol{\mu}}$ in step 9 requires N^2 operations, and is performed for L sequences $u\boldsymbol{\mu} \in \mathtt{U}_t$. These computations, in addition to those of the PruneStep call in step 7, are repeated for h iterations of the main loop over t. Thus, the overall complexity of Algorithm 1 is $O(h^2 n_u M_x[N^2 + LN] + hLN^2)$.

We further remark that the algorithm for pruning and computing the $\mathbf{d}_{\boldsymbol{\mu}}$ vectors only needs to be performed once, and may be done offline. The online implementation of a control policy then only needs to compute the cost ${\mathbf{d}_{\boldsymbol{\mu}}}^T \boldsymbol{\xi}_0$ for each sequence $\boldsymbol{\mu} \in \mathtt{U}_0$. Thus, the online complexity of evaluating $\widehat{J}_0^{\mathrm{K}}(\boldsymbol{\xi}_0)$ is $O(LN)$.

For reference, we may compare with the computational complexity of the full cost-to-go (4.7). The offline computation of all vectors $\mathbf{d}(\boldsymbol{\mu})$ may be done via (4.9), which requires computation of the product $A(u)^T \mathbf{d}(\boldsymbol{\mu})$ for all sequences $u\boldsymbol{\mu} \in \mathcal{U}^{h-t}$, for each $t \in \{h-1, \ldots, 0\}$. Thus, the total number of operations is $\sum_{t=0}^{h-1} {n_u}^{h-t} N^2$, which has complexity $O({n_u}^h N^2)$. The online complexity is $O({n_u}^h N)$, as it is the same as the pruned version but with exponentially many sequences.

The offline and online complexities of the pruned and unpruned cost-to-go functions are summarized in Table 4.1.

4.2.4 Numerical Results

Effective control of the MultiPong and Assault systems require being able to look many time steps into the future, which our pruning algorithm is crucial for. Now that we are no longer limited in the horizon lengths we may consider, we showcase several experimental results for control of these systems.

MultiPong

We first consider the MultiPong system, initialized with discretized states as in (3.1). We consider initializations with various values for m_x, but we fix $m_y = 3$, so that the ball's y-velocity is always much faster than the paddle can move, necessitating prediction of its future trajectory. Our objective is to keep the balls to the left of the paddle as much as possible.

Following the same process as in Example 3.5, we define two dictionaries $\mathbf{\Psi}_{\text{sep}}$ and $\mathbf{\Psi}_{\text{prod}}$, and learn the corresponding matrices $A(u), C(u)$ via EDMD. We do this learning only for the system with one ball, as our entity-based framework allows them to re-use them for systems with multiple balls. For each Koopman model, we consider the optimal control problem optimal control problem (4.2) with a cost function that counts the number of balls to the right of the paddle, and with a horizon $h = 20$. With three possible inputs, there are a total $3^{20} \approx 3.5$ billion possible input sequences. As such, we construct pruned control policies via Algorithm 1, with capacity $L = 10{,}000$, with $M_r = 5{,}000$ random sequences per iteration, with $M_b = 25$ best sequences per test state, and with the initial state set $\mathtt{X}_0$ taken as a set of 10 random initial states. Due to the randomness of the exploration phase, we construct ten such policies for each Koopman model, and we examine their mean performance.

For reference, we compare our Koopman-based policies with two non-Koopman poli-

cies that we now describe. The first is a "greedy" policy, which moves the paddle toward the closest ball's current y-coordinate. If all balls are to the right of the paddle, it instead moves away from that y-coordinate (to avoid deflecting the ball back toward the right). The second is a "predictive" policy that instead moves the paddle toward the position where the closest ball will cross the paddle's x-coordinate, which is determined exactly via full knowledge of the nonlinear dynamics. We remark that the predictive policy is nearly optimal for one ball, but is suboptimal for multiple balls since the optimal prioritization among the balls is nontrivial.

The results are shown in Figure 4.4. As we might expect based on Example 3.5, the use of product densities is crucial for effective control of the system. With only separate densities, we achieve performance comparable to the greedy policy. With product densities, the Koopman approach matches the predictive policy for one ball, and beats it for three balls.

We also evaluate the effect of varying the controller capacity L. For systems with $m_x \in \{5, 10, 15\}$, and using the Koopman model with product densities, we test control policies with a range of capacities from $L = 1{,}000$ to $L = 100{,}000$ (while also varying M_b and M_r proportionally to ensure that (4.10) remains satisfied). The results are shown in Figure 4.5. Even with relatively small controllers, we are already able to achieve good performance. The system with three balls and $m_x = 10$ is where we see the most substantial improvement for larger controllers, but even there we see substantial diminishing returns for capacities larger than $L = 10{,}000$.

Assault

We now consider the game Assault. Since the system is only partially observable via the current game screen, we use time-delayed observables as in [35]. Letting $\mathbf{s}_t$ denote the screen at the current time step, we allow observables to depend on both the current

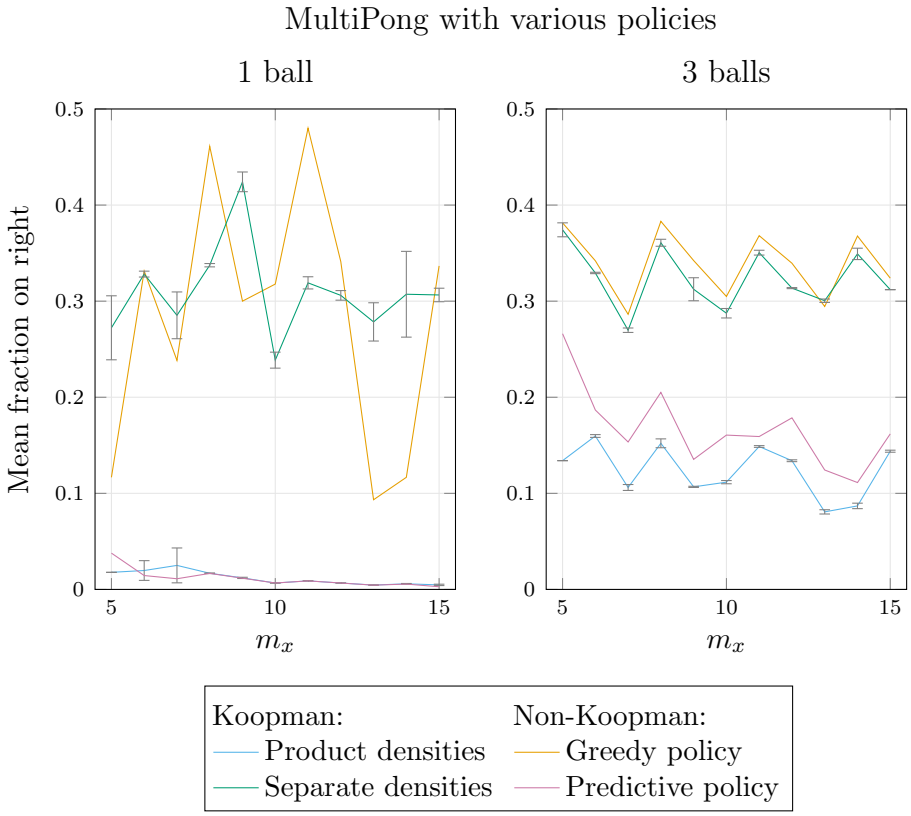

Figure 4.4: Performance of different control policies for MultiPong, measured in terms of the mean fraction of time that balls spend to the right of the paddle (lower is better). The parameter m_x used for the discretized initial states in (3.1) is varied, and the size of the ball density is varied proportionally. The overall dictionary sizes range from $N = 72$ to $N = 192$ for separate densities, and from $N = 720$ to $N = 2160$ for product densities. Error bars denote 95% confidence intervals.

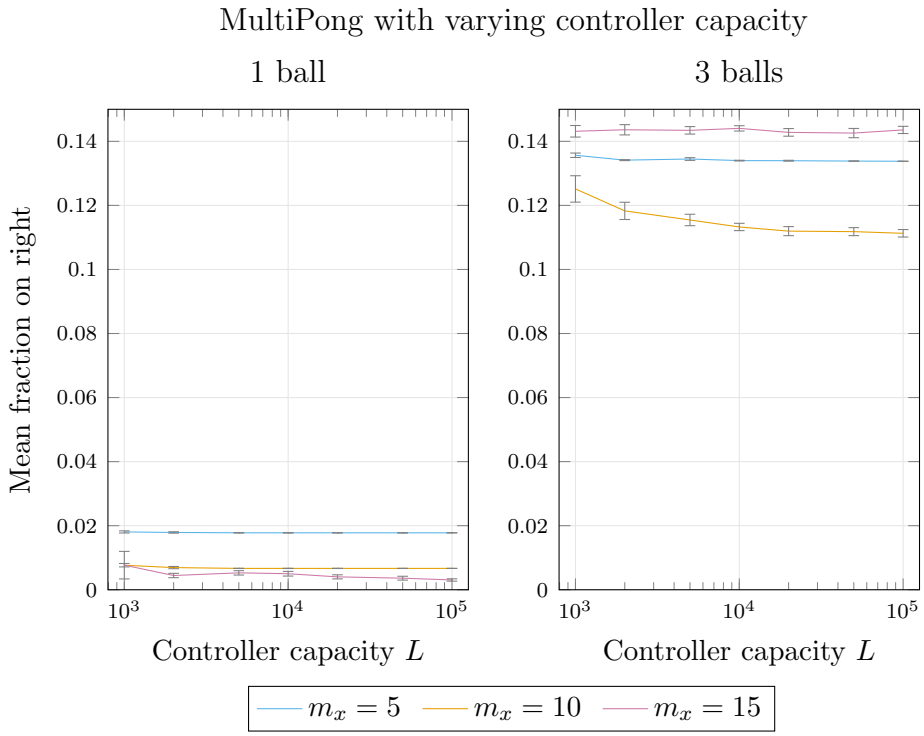

Figure 4.5: Performance of Koopman-based control policies with product densities, constructed via Algorithm 1 for a range of capacities L.

screen $\mathbf{s}_t$ and the previous screen $\mathbf{s}_{t-1}$. We define two dictionaries, $\mathbf{\Psi}_{\text{sep}}$ consisting of only separate densities and $\mathbf{\Psi}_{\text{comb}}$ consisting of a combination of separate and product densities, as

$$\mathbf{\Psi}_{\text{sep}}(\mathbf{s}_t, \mathbf{s}_{t-1}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{s}_t) \\ \mathbf{\Phi}_2(\mathbf{s}_t) \\ \mathbf{\Phi}_3(\mathbf{s}_t) \\ \mathbf{\Phi}_4(\mathbf{s}_t) \\ \mathbf{\Phi}_5(\mathbf{s}_t) \\ \hdashline \mathbf{\Phi}_1(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_3(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_4(\mathbf{s}_{t-1}) \end{bmatrix}, \qquad \mathbf{\Psi}_{\text{comb}}(\mathbf{s}_t, \mathbf{s}_{t-1}) = \begin{bmatrix} \mathbf{\Phi}_1(\mathbf{s}_t) \\ \mathbf{\Phi}_2(\mathbf{s}_t) \\ \mathbf{\Phi}_3(\mathbf{s}_t) \\ \mathbf{\Phi}_4(\mathbf{s}_t) \\ \mathbf{\Phi}_5(\mathbf{s}_t) \\ \hdashline \mathbf{\Phi}_1(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_2(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_3(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_4(\mathbf{s}_{t-1}) \\ \mathbf{\Phi}_5(\mathbf{s}_{t-1}) \\ \hdashline \mathbf{\Phi}_{1\otimes 3}(\mathbf{s}_t) \\ \mathbf{\Phi}_{1\otimes 4}(\mathbf{s}_t) \\ \mathbf{\Phi}_{2\otimes 3}(\mathbf{s}_t) \end{bmatrix}.$$

Note that $\mathbf{\Psi}_{\text{comb}}$ includes product densities for only three pairs of entity classes, rather than for all ten pairs, to keep the size of the dictionary manageable. The inclusion of the product density $\mathbf{\Phi}_{1\otimes 4}$ (player ship, enemy fire) is needed to determine when the player is hit, and vice versa for $\mathbf{\Phi}_{2\otimes 3}$ (enemy ships, player fire). The product density $\mathbf{\Phi}_{1\otimes 3}$ (player ship, enemy ships) is then needed to know where to insert new fire into $\mathbf{\Phi}_{1\otimes 4}$ and $\mathbf{\Phi}_{2\otimes 3}$.

Both dictionaries use K-means clusterings to choose the basis centers for each entity class, with differing numbers of centers for each class as listed in Table 4.2. The overall dictionary sizes are $N = 616$ for $\mathbf{\Psi}_{\text{sep}}$, and $N = 1{,}032$ for $\mathbf{\Psi}_{\text{comb}}$. We specify an output function $\mathbf{g}(x) = [k(x)\ r(x)]^T$, where $k(x)$ is an indicator of whether the player was killed in the current time step, and $r(x)$ returns the number of points scored in the current time step. The $A(u)$ and C matrices for each dictionary are learned via EDMD (2.14), for data snapshots (2.13) generated via random play. We use 500 episodes (662,668 snapshots) for $\mathbf{\Psi}_{\text{sep}}$, and 1,000 episodes (1,320,728 snapshots) for $\mathbf{\Psi}_{\text{comb}}$.

Table 4.2: Numbers of basis centers for each entity class for $\mathbf{\Psi}_{\text{sep}}$ and $\mathbf{\Psi}_{\text{comb}}$.

i	Entity class	N_i for $\mathbf{\Psi}_{\text{sep}}$	N_i for $\mathbf{\Psi}_{\text{comb}}$
1	Player ship	50	12
2	Player fire	100	24
3	Enemy ships	185	20
4	Enemy fire	15	12
5	Attack gauge	16	16

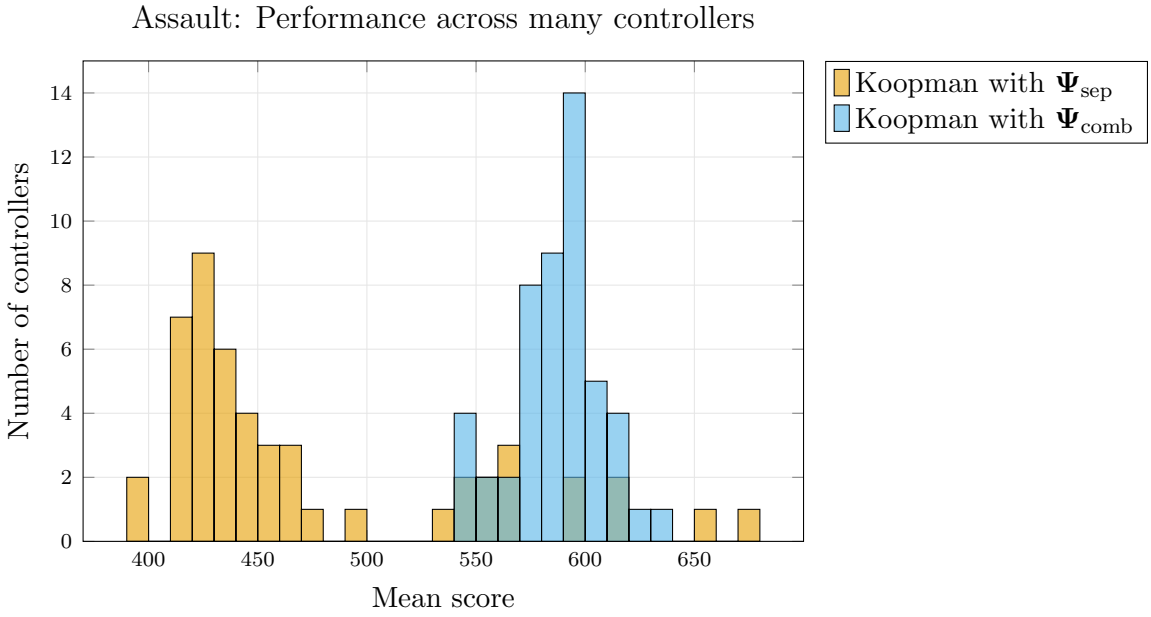

Figure 4.6: A histogram of the performance of 50 control policies for each Koopman model, computed as the mean score achieved across 50 episodes.

We consider the optimal control problem (4.3) with horizon $h = 50$ and cost function $\gamma(x) = \alpha k(x) - \beta r(x)$ for chosen weights $\alpha, \beta > 0$. Using Algorithm 1 with capacity $L = 10{,}000$, we construct 50 different control policies for each Koopman model, and we test each policy across 50 episodes. We find that we achieve good control policies more consistently with the Koopman model for $\mathbf{\Psi}_{\text{comb}}$, as shown in Figure 4.6.

Interestingly, the best controllers obtained from each Koopman model appear to achieve comparable performance, which we confirm by testing each on a larger sample of 1,000 episodes. The limiting factor for both is a game mechanic that first occurs in

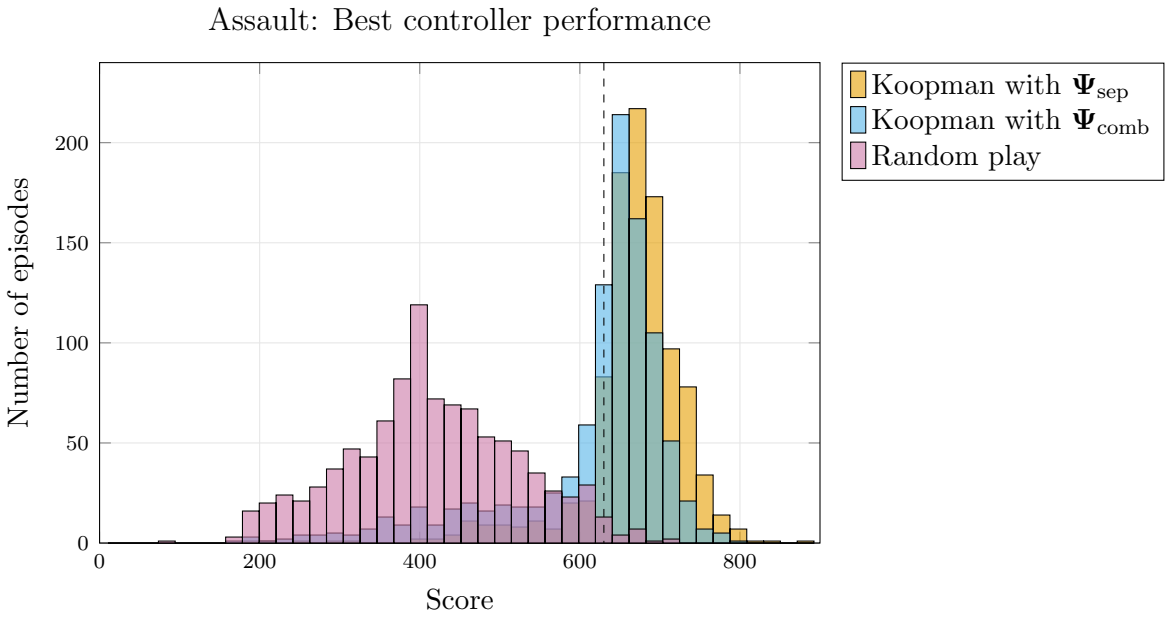

Figure 4.7: A histogram of the scores achieved across 1,000 episodes by the best controller for each Koopman model, alongside scores from 1,000 episodes of random play. The vertical dashed line at the score 630 indicates where the fourth level of the game begins, which introduces a new mechanic that limits our controller performance.

the fourth level of the game, which is reached when at a score of 630. As seen from Figure 4.7, both controllers reach the fourth level most of the time, whereas it is almost never reached by random play. The controllers are thus unable to react to this new mechanic, since it is absent from the training data their models are learned from.

4.3 State Constraints

We now incorporate a set of constraints on the final state x_h, expressed in vector form by a nonlinear function $\boldsymbol{\zeta} : \mathcal{X} \to \mathbb{R}^{n_c}$ and vector $\mathbf{z} \in \mathbb{R}^{n_c}$ for which we wish to enforce that the inequality $\boldsymbol{\zeta}(x_h) \leq \mathbf{z}$ is satisfied elementwise. Thus, we are interested in

obtaining a solution to the following constrained optimal control problem.

$$\min_{\boldsymbol{\mu}_0^h \in \mathcal{U}^h} \quad \gamma_h(x_h) + \sum_{t=0}^{h-1} \gamma_t(x_t, u_t), \tag{4.12a}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t), \quad t \in \{0, \ldots, h-1\}, \tag{4.12b}$$

$$\boldsymbol{\zeta}(x_h) \leq \mathbf{z}. \tag{4.12c}$$

The associated optimal cost-to-go for this constrained problem is given by the function $J_t^{\text{NL-C}} : \mathcal{X} \to \mathbb{R} \cup \{\infty\}$, defined by

$$J_t^{\text{NL-C}}(x_t) := \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \quad \gamma_h(x_h) + \sum_{\tau=t}^{h-1} \gamma_\tau(x_\tau, u_\tau) \tag{4.13a}$$

$$\text{s.t.} \quad x_{\tau+1} = f(x_\tau, u_\tau), \quad \tau \in \{t, \ldots, h-1\}, \tag{4.13b}$$

$$\boldsymbol{\zeta}(x_h) \leq \mathbf{z}, \tag{4.13c}$$

with the convention that $J_t^{\text{NL-C}}(x_t) = \infty$ if the constraints are infeasible.

We now consider both the cost functions and the constraint functions to be outputs of the system

$$x^+ = f(x, u), \tag{4.14a}$$

$$\mathbf{y} = \mathbf{g}(x, u) := \begin{bmatrix} \gamma_0(x,u) & \cdots & \gamma_{h-1}(x,u) & \gamma_h(x) & \boldsymbol{\zeta}(x)^T \end{bmatrix}^T. \tag{4.14b}$$

If we can obtain an exact Koopman representation for this system, then we can equivalently represent the constrained nonlinear problem (4.12) by a constrained linear problem.

Proposition 4.5. *Suppose* $(\boldsymbol{\Psi}, A, C)$ *is an exact Koopman representation for* (4.5) *where*

$$C(u) = \begin{bmatrix} —\mathbf{c}_0(u)^T— \\ \vdots \\ —\mathbf{c}_{h-1}(u)^T— \\ —\mathbf{c}_h^T— \\ Z \end{bmatrix}, \qquad \forall u \in \mathcal{U},$$

for mappings $\mathbf{c}_t : \mathcal{U} \to \mathbb{R}^N,\ t \in \{0, \dots, h-1\}$, *and for* $\mathbf{c}_h \in \mathbb{R}^N$ *and* $Z \in \mathbb{R}^{n_c \times N}$. *For* $t \in \{0, \dots, h\}$, *define the function* $J_t^{\text{K-C}} : \mathbb{R}^N \to \mathbb{R} \cup \{\infty\}$ *by*

$$J_t^{\text{K-C}}(\boldsymbol{\xi}_t) := \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \quad \mathbf{c}_h^T \boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \boldsymbol{\xi}_\tau \tag{4.15a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_{\tau+1} = A(u_\tau)\boldsymbol{\xi}_\tau, \quad \tau \in \{t, \dots, h-1\}, \tag{4.15b}$$

$$Z\boldsymbol{\xi}_h \le \mathbf{z}. \tag{4.15c}$$

Then

$$J_t^{\text{NL-C}}(x_t) = J_t^{\text{K-C}}(\boldsymbol{\Psi}(x_t)), \qquad \forall x_t \in \mathcal{X},\ \forall t \in \{0, \dots, h\}.$$

Proof: Fix $x_t \in \mathcal{X}$ and $\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}$, and let $\boldsymbol{\xi}_t = \boldsymbol{\Psi}(x_t)$. The equivalence of the objectives (4.13a) and (4.15a) follows directly by the proof of Proposition 4.1. We now also have

$$\boldsymbol{\zeta}(x_h) = Z\boldsymbol{\Psi}(x_h) = Z\boldsymbol{\xi}_h,$$

so that the constraints (4.13c) and (4.15c) are equivalent. Since both the objectives and constraints are equivalent for any $x_t \in \mathcal{X}$ and $\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}$, then we conclude that $J_t^{\text{NL-C}}(x_t) = J_t^{\text{K-C}}(\boldsymbol{\Psi}(x_t))$ as desired. ■

We obtain the following result on the constrained cost-to-go, similar to that of Theorem 4.2 for the unconstrained case.

Theorem 4.6. *The cost-to-go function* $J_t^{\text{K-C}}$ *as defined by* (4.15) *is piecewise linear and*

given by

$$J_t^{\text{K-C}}(\boldsymbol{\xi}_t) = \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \quad \mathbf{d}(\boldsymbol{\mu}_t^h)^T \boldsymbol{\xi}_t, \tag{4.16a}$$

$$\text{s.t.} \quad W(\boldsymbol{\mu}_t^h)\boldsymbol{\xi}_t \leq \mathbf{z} \tag{4.16b}$$

where $\mathbf{d}$ *is defined as in* (4.8) *and* $W : \mathcal{U}^* \to \mathbb{R}^{n_c \times N}$ *is defined as*

$$W(\boldsymbol{\mu}_t^h) := Z\bar{A}(\boldsymbol{\mu}_t^h). \tag{4.17}$$

Proof: The equivalence of the objectives (4.15a) and (4.16a) follows directly from the proof of Theorem 4.2. We are left only to prove that the constraints (4.15c) and (4.16b) are equivalent. This follows directly from the fact that $\boldsymbol{\xi}_h = \bar{A}(\boldsymbol{\mu}_t^h)\boldsymbol{\xi}_t$, and thus

$$Z\boldsymbol{\xi}_h = Z\bar{A}(\boldsymbol{\mu}_t^h)\boldsymbol{\xi}_t = W(\boldsymbol{\mu}_t^h)\boldsymbol{\xi}_t.$$

■

To consider the constrained problem with longer horizons, we require a modified version of the dynamic pruning algorithm In Algorithm 1, we insert the step

$$W_\epsilon \leftarrow Z$$

after step 2, and we insert the step

$$W_{u\boldsymbol{\mu}} \leftarrow W_{\boldsymbol{\mu}} A(u)$$

after step 9 (but still inside the loop). The definition of the set $\mathtt{D}_t$ in step 10 is changed

to

$$\mathtt{D}_t \leftarrow \{(\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}) : \boldsymbol{\mu} \in \mathtt{U}_t\}.$$

An additional consideration is whether we should use the constraints during the exploitation phase of the PruneStep function. This may be achieved by modifying the definition of $J_{u\boldsymbol{\mu}}$ in step 8 of Algorithm 2 to be

$$J_{u\boldsymbol{\mu}} \leftarrow \begin{cases} {\mathbf{d}_{\boldsymbol{\mu}}}^T \boldsymbol{\xi}^u_{j,k} + c^u_{j,k}, & W_{\boldsymbol{\mu}} \boldsymbol{\xi}^u_{j,k} \leq \mathbf{z}, \\ \infty, & \text{otherwise.} \end{cases} \tag{4.18}$$

In this way, the sequences $u\boldsymbol{\mu} \in \mathcal{U} \cdot \mathtt{U}_{t+1}$ that achieve the M_b lowest costs $J_{u\boldsymbol{\mu}}$ will consist only of those for which the constraint is satisfied from the state $\boldsymbol{\xi}_{j,k}$ (unless fewer than M_b sequences satisfy the constraint). However, this modification is computationally costly as it adds a matrix-vector multiplication to the innermost loop. As such, it is likely preferable to do the pruning without constraints, unless there is reason to believe that constrained pruning would lead to substantially better performance.

In either case, the end result of our modified Algorithm 1 is a set of sequences $\mathtt{U}_0 \in \mathcal{U}^h$ and the corresponding $\mathtt{D}_0 = \{(\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}) : \boldsymbol{\mu} \in \mathtt{U}_0\}$, with which we approximate the constrained cost-to-go (4.16) as

$$\begin{aligned} \widehat{J}_0^{\text{K-C}}(\boldsymbol{\xi}_0) = \min_{\boldsymbol{\mu} \in \mathcal{U}^h} \quad & {\mathbf{d}_{\boldsymbol{\mu}}}^T \boldsymbol{\xi}_0, \\ \text{s.t.} \quad & W_{\boldsymbol{\mu}} \boldsymbol{\xi}_t \leq \mathbf{z}. \end{aligned}$$

We now demonstrate the use of constraints via the MultiPong system with multiple paddles.

Example 4.7. We consider the MultiPong system, initialized with discretized states as

in (3.1) with $m_x = 5$ and $m_y = 3$, and with multiple balls and paddles. Since it is disadvantageous for multiple paddles to occupy the same position, we implement constraints such that each position should have at most one paddle. We solve this constrained problem with a horizon $h = 12$ via Koopman models with product density observables that are each learned via a system with one ball and one paddle. We compare the performance of unconstrained and constrained control policies. For the constrained policies, we consider both constrained pruning via the modification in (4.18), and unconstrained pruning.

The results are shown in Figure 4.8. We see that constrained policies always perform better than the unconstrained policies, especially when there are at least as many paddles as balls. The use of constraints during pruning has negligible impact on the resulting performance, indicating that it is preferable to ignore constraints during pruning to save on computation.

4.4 Robust Control

We recall that we generally work with Koopman representations that are not exact, i.e. for which there is some nonzero error in our predictions of future lifted states. It may then be desirable to incorporate some form of robustness into our controller design. In addition to prediction errors, we may also consider error in the initial lifted state, such as from measurement noise. We consider a version of our optimal control problem in which we guard against these errors, up to specified bounds on their p-norm for some $p \in [1, \infty]$.

Let $\hat{\varepsilon} > 0$ denote a bound on the initial state error, and let $\varepsilon : \mathcal{U} \to \mathbb{R}_{>0}$ denote an

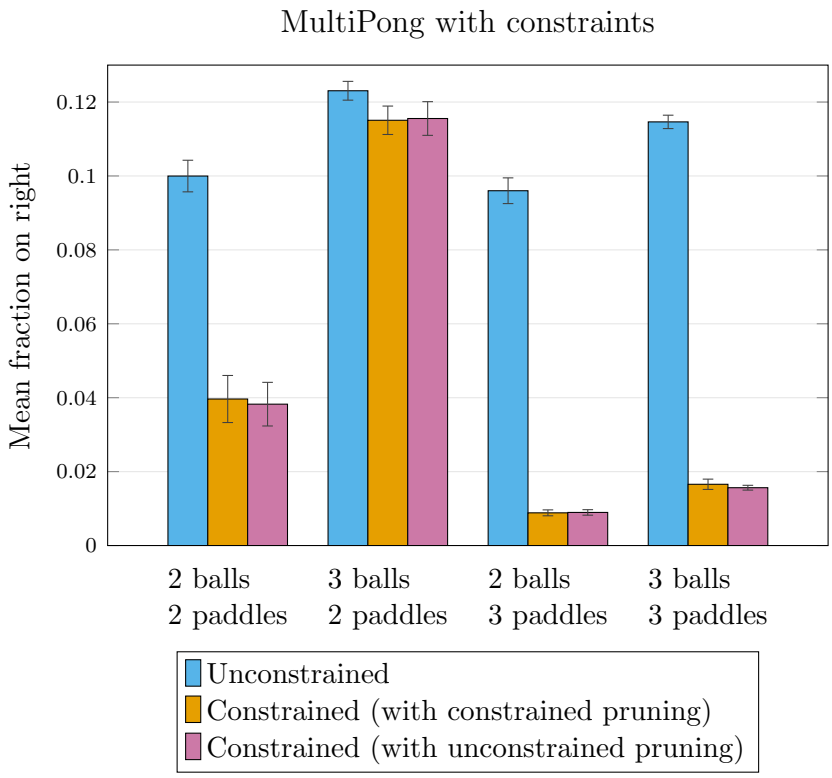

Figure 4.8: Performance of constrained and unconstrained Koopman-based control policies for MultiPong systems with multiple balls and multiple paddles. The constrained policies enforce that we never move two paddles into the same position at the same time.

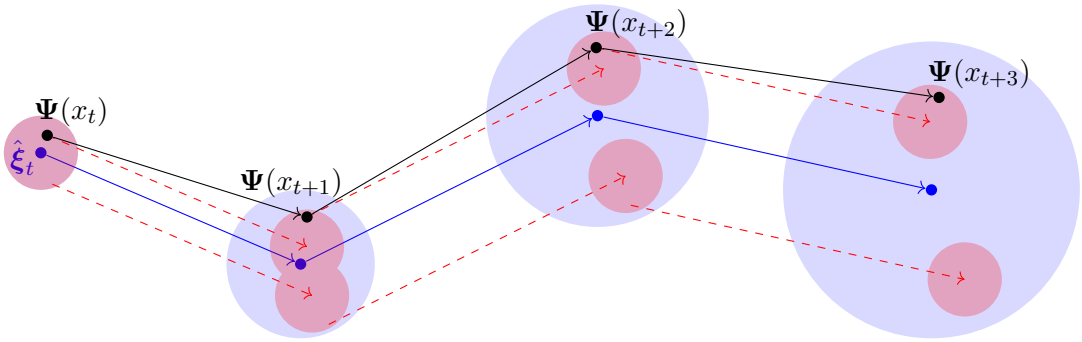

Figure 4.9: A visualization of the uncertain trajectory set described in (4.20). The blue state trajectory represents the nominal prediction of the Koopman model from $\hat{\boldsymbol{\xi}}_t$, whereas the red represents two possible sequences and the uncertainty bounds at each time. The blue regions show the cumulative uncertainty. The true state trajectory is unknown, but will lie in the set if the bounds (4.19) are satisfied.

input-dependent bound on the prediction error, such that

$$\|\boldsymbol{\Psi}(x_t) - \hat{\boldsymbol{\xi}}_t\|_p \leq \hat{\varepsilon}, \qquad \|\boldsymbol{\Psi}(x_{\tau+1}) - A(u_\tau)\boldsymbol{\Psi}(x_\tau)\|_p \leq \varepsilon(u_\tau), \ \forall \tau \geq t \tag{4.19}$$

where $\hat{\boldsymbol{\xi}}_t$ denotes our (possibly inaccurate) estimate of $\boldsymbol{\Psi}(x_t)$. Similar bounded uncertainties are employed in robust tube-based control approaches such as in [70]. When an input sequence $\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}$ is applied from the state $x_t \in \mathcal{X}$, the set of all possible lifted state trajectories that satisfy these bounds is given by

$$\mathcal{T}(\hat{\boldsymbol{\xi}}_t, \boldsymbol{\mu}_t^h) := \Big\{(\boldsymbol{\xi}_t, \ldots, \boldsymbol{\xi}_h) \in (\mathbb{R}^N)^{h-t+1} \ : \ \|\boldsymbol{\xi}_t - \hat{\boldsymbol{\xi}}_t\|_p \leq \hat{\varepsilon}, \tag{4.20}$$
$$\|\boldsymbol{\xi}_{\tau+1} - A(u_\tau)\boldsymbol{\xi}_\tau\|_p \leq \varepsilon(u_\tau) \ \forall \tau \in \{t, \ldots, h-1\}\Big\}.$$

These uncertainty constraints are illustrated in Figure 4.9.

We now wish to find an input sequence $\boldsymbol{\mu}_t^h$ that minimizes the worst-case cost among trajectories in this set, while ensuring that the final state constraint is satisfied. This is equivalent to identifying the argmin of the robust cost-to-go function $J_t^{\text{K-R}} : \mathcal{X} \to$

$\mathbb{R} \cup \{\infty\}$, defined by

$$J_t^{\text{K-R}}(x_t) := \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \quad \max_{(\boldsymbol{\xi}_t, \ldots, \boldsymbol{\xi}_h) \in \mathcal{T}(x_t, \boldsymbol{\mu}_t^h)} \mathbf{c}_h^T \boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \boldsymbol{\xi}_\tau \tag{4.21a}$$

$$\text{s.t.} \quad Z\boldsymbol{\xi}_h \leq \mathbf{z}, \quad \forall (\boldsymbol{\xi}_t, \ldots, \boldsymbol{\xi}_h) \in \mathcal{T}(x_t, \boldsymbol{\mu}_t^h). \tag{4.21b}$$

It turns out that this function has a piecewise affine structure similar to that of the unconstrained case, as described by the following result.

Theorem 4.8. *The robust cost-to-go $J_t^{\text{K-R}}$ is piecewise affine with respect to the estimated lifted state $\hat{\boldsymbol{\xi}}_t$, and is given by*

$$J_t^{\text{K-C}}(\hat{\boldsymbol{\xi}}_t) = \min_{\boldsymbol{\mu}_t^h \in \mathcal{U}^{h-t}} \quad \mathbf{d}(\boldsymbol{\mu}_t^h)^T \hat{\boldsymbol{\xi}}_t + \beta(\boldsymbol{\mu}_t^h), \tag{4.22a}$$

$$\text{s.t.} \quad W(\boldsymbol{\mu}_t^h)\hat{\boldsymbol{\xi}}_t \leq \boldsymbol{\omega}(\boldsymbol{\mu}_t^h), \tag{4.22b}$$

where $\mathbf{d}$ *and* W *are as defined in* (4.8) *and* (4.17)*, and* $\beta : \mathcal{U}^* \to \mathbb{R}$ *and* $\boldsymbol{\omega} : \mathcal{U}^* \to \mathbb{R}^{n_c}$ *are defined as*

$$\beta(\boldsymbol{\mu}_t^h) := \hat{\varepsilon}\|\mathbf{d}(\boldsymbol{\mu}_t^h)\|_q + \sum_{s=t}^{h-1} \varepsilon(u_s)\|\mathbf{d}(\boldsymbol{\mu}_{s+1}^h)\|_q,$$

$$\boldsymbol{\omega}(\boldsymbol{\mu}_t^h) := \mathbf{z} - \hat{\varepsilon}\boldsymbol{\rho}_q(W(\boldsymbol{\mu}_t^h)) - \sum_{s=t}^{h-1} \varepsilon(u_s)\boldsymbol{\rho}_q(W(\boldsymbol{\mu}_{s+1}^h)),$$

where $q \in [1, \infty]$ *is such that* $\frac{1}{p} + \frac{1}{q} = 1$*, and* $\boldsymbol{\rho}_q : \mathbb{R}^{n_c \times N} \to \mathbb{R}^{n_c}$ *denotes a "row-wise q-norm" defined by*

$$\boldsymbol{\rho}_q\left(\begin{bmatrix} —\mathbf{w}_1^T— \\ \vdots \\ —\mathbf{w}_{n_c}^T— \end{bmatrix}\right) := \begin{bmatrix} \|\mathbf{w}_1\|_q \\ \vdots \\ \|\mathbf{w}_{n_c}\|_q \end{bmatrix}.$$

Proof: We first note that a lifted state trajectory $(\boldsymbol{\xi}_t, \ldots, \boldsymbol{\xi}_h)$ belongs to the set $\mathcal{T}(x_t, \boldsymbol{\mu}_t^h)$ if and only if there exist vectors $\hat{\mathbf{v}}, \mathbf{v}_t, \ldots, \mathbf{v}_{h-1} \in \mathbb{R}^N$ with $\|\hat{\mathbf{v}}\|_p \leq \hat{\varepsilon}$ and

$\|\mathbf{v}_s\|_p \leq \varepsilon(u_s)$, $s \in \{t, \ldots, h-1\}$ such that

$$\begin{aligned} \boldsymbol{\xi}_t &= \hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}, \\ \boldsymbol{\xi}_{s+1} &= A(u_s)\boldsymbol{\xi}_s + \mathbf{v}_s, \quad \forall s \in \{t, \ldots, h-1\}. \end{aligned}$$

Recursively iterating these expressions, we then have

$$\boldsymbol{\xi}_\tau = \bar{A}(\boldsymbol{\mu}_t^\tau)[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{\tau-1} \bar{A}(\boldsymbol{\mu}_{s+1}^\tau)\mathbf{v}_s, \qquad \forall \tau \in \{t, \ldots, h\}. \tag{4.23}$$

Substituting this expression (with $\tau = h$) into the left side of the vector inequality in (4.21b) yields

$$\begin{aligned} Z\boldsymbol{\xi}_h &= Z\bar{A}(\boldsymbol{\mu}_t^h)[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{\tau-1} Z\bar{A}(\boldsymbol{\mu}_{s+1}^h)\mathbf{v}_s \\ &= W(\boldsymbol{\mu}_t^h)[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{\tau-1} W(\boldsymbol{\mu}_{s+1}^h)\mathbf{v}_s. \end{aligned}$$

To guarantee that the constraint is satisfied, we must ensure that the worst-case value of each component remains bounded by the corresponding limit in the vector $\mathbf{z}$. Thus, for each $i \in \{1, \ldots, n_c\}$, we must enforce that

$$\max_{\substack{\|\hat{\mathbf{v}}\|_p \leq \hat{\varepsilon} \\ \|\mathbf{v}_s\|_p \leq \varepsilon(u_s),\, s \in \{t, \ldots, h-1\}}} \mathbf{w}_i(\boldsymbol{\mu}_t^h)^T[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{\tau-1} \mathbf{w}_i(\boldsymbol{\mu}_{s+1}^h)^T\mathbf{v}_s \leq z_i, \tag{4.24}$$

where $\mathbf{w}_i(\boldsymbol{\mu}_t^h)^T$ is the ith row of $W(\boldsymbol{\mu}_t^h)$, and z_i is the ith component of $\mathbf{z}$.

Since the vectors $\hat{\mathbf{v}}, \mathbf{v}_t, \ldots, \mathbf{v}_{h-1}$ are independent and appear in decoupled terms that are added together, then the maximization in (4.24) is achieved by maximizing each term individually. We note that, if $p, q \in [1, \infty]$ are such $\frac{1}{p} + \frac{1}{q} = 1$, then the norms $\|\cdot\|_p$ and

$\|\cdot\|_q$ are dual to each other. For any vector $\mathbf{w}$ and scalar $\alpha > 0$, we then have that

$$\max_{\|\mathbf{v}\|_p \leq \alpha} \mathbf{w}^T \mathbf{v} = \alpha \|\mathbf{w}\|_q.$$

Using this fact, we may write (4.24) as

$$\mathbf{w}_i(\boldsymbol{\mu}_t^h)^T \hat{\boldsymbol{\xi}}_t + \hat{\varepsilon}\|\mathbf{w}_i(\boldsymbol{\mu}_t^h)\|_q + \sum_{s=t}^{\tau-1} \varepsilon(u_s)\|\mathbf{w}_i(\boldsymbol{\mu}_{s+1}^h)\|_q \leq z_i.$$

Rearranging this expression yields the ith constraint of (4.22b) as desired.

By a similar argument, substituting (4.23) into (4.21a) yields

$$\begin{aligned}
&\mathbf{c}_h^T \boldsymbol{\xi}_h + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \boldsymbol{\xi}_\tau \\
&\quad = \mathbf{c}_h^T \left[\bar{A}(\boldsymbol{\mu}_t^h)[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{h-1} \bar{A}(\boldsymbol{\mu}_{s+1}^h)\mathbf{v}_s \right] \\
&\qquad\qquad + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \left[\bar{A}(\boldsymbol{\mu}_t^\tau)[\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{\tau-1} \bar{A}(\boldsymbol{\mu}_{s+1}^\tau)\mathbf{v}_s \right] \\
&\quad = \left[\mathbf{c}_h^T \bar{A}(\boldsymbol{\mu}_t^h) + \sum_{\tau=t}^{h-1} \mathbf{c}_\tau(u_\tau)^T \bar{A}(\boldsymbol{\mu}_t^\tau) \right] [\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] \\
&\qquad\qquad + \left[\sum_{s=t}^{h-1} \mathbf{c}_h^T \bar{A}(\boldsymbol{\mu}_{s+1}^h) + \sum_{\tau=t}^{h-1} \sum_{s=t}^{\tau-1} \mathbf{c}_\tau(u_\tau)^T \bar{A}(\boldsymbol{\mu}_{s+1}^\tau) \right] \mathbf{v}_s \\
&\quad = \left[\bar{A}(\boldsymbol{\mu}_t^h)^T \mathbf{c}_h + \sum_{\tau=t}^{h-1} \bar{A}(\boldsymbol{\mu}_t^\tau)^T \mathbf{c}_\tau(u_\tau) \right]^T [\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] \\
&\qquad\qquad + \sum_{s=t}^{h-1} \left[\bar{A}(\boldsymbol{\mu}_{s+1}^h)^T \mathbf{c}_h + \sum_{\tau=s+1}^{h-1} \bar{A}(\boldsymbol{\mu}_{s+1}^\tau)^T \mathbf{c}_\tau(u_\tau) \right]^T \mathbf{v}_s, \\
&\quad = \mathbf{d}(\boldsymbol{\mu}_t^h)^T [\hat{\boldsymbol{\xi}}_t + \hat{\mathbf{v}}] + \sum_{s=t}^{h-1} \mathbf{d}(\boldsymbol{\mu}_{s+1}^h)^T \mathbf{v}_s.
\end{aligned}$$

Maximizing this expression with respect to the bounds on the vectors $\hat{\mathbf{v}}, \mathbf{v}_t, \ldots, \mathbf{v}_{h-1}$, we obtain (4.22a) as desired. This completes the proof. ■

The dynamic pruning algorithm may be further modified for the robust problem as follows. In Algorithm 1, we insert the steps

$$\beta_\epsilon \leftarrow \hat{\varepsilon}\|\mathbf{c}_h\|_q$$
$$\boldsymbol{\omega}_\epsilon \leftarrow \mathbf{z} - \hat{\varepsilon}\boldsymbol{\rho}_q(Z)$$

after step 2, and we insert the steps

$$\beta_{u\boldsymbol{\mu}} \leftarrow \beta_{\boldsymbol{\mu}} + \hat{\varepsilon}\|\mathbf{d}_{u\boldsymbol{\mu}}\|_q + [\varepsilon(u) - \hat{\varepsilon}]\|\mathbf{d}_{\boldsymbol{\mu}}\|_q$$
$$\boldsymbol{\omega}_{u\boldsymbol{\mu}} \leftarrow \boldsymbol{\omega}_{\boldsymbol{\mu}} - \hat{\varepsilon}\boldsymbol{\rho}_q(W_{u\boldsymbol{\mu}}) - [\varepsilon(u) - \hat{\varepsilon}]\boldsymbol{\rho}_q(W_{\boldsymbol{\mu}})$$

after step 9 (but still inside the loop). The definition of the set $\mathtt{D}_t$ in step 10 is changed to

$$\mathtt{D}_t \leftarrow \{(\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}, \beta_{\boldsymbol{\mu}}, \boldsymbol{\omega}_{\boldsymbol{\mu}}) : \boldsymbol{\mu} \in \mathtt{U}_t\}.$$

With this modified algorithm, we then approximate the robust worst-case cost-to-go (4.22) as

$$\widehat{J}_0^{\text{K-R}}(\boldsymbol{\xi}_0) = \min_{\boldsymbol{\mu} \in \mathcal{U}^h} \quad \mathbf{d}_{\boldsymbol{\mu}}{}^T \boldsymbol{\xi}_0 + \beta_{\boldsymbol{\mu}},$$
$$\text{s.t.} \quad W_{\boldsymbol{\mu}} \boldsymbol{\xi}_0 \leq \boldsymbol{\omega}_{\boldsymbol{\mu}}.$$

Example 4.9. Consider the MultiPong system with the perturbed initialization (3.2) with $m_x = 5$ and $m_y = 3$. The results for the non-robust control policies are shown in Figure 4.10. The Koopman controllers remain effective for small δ perturbations, but the performance degrades as δ grows large.

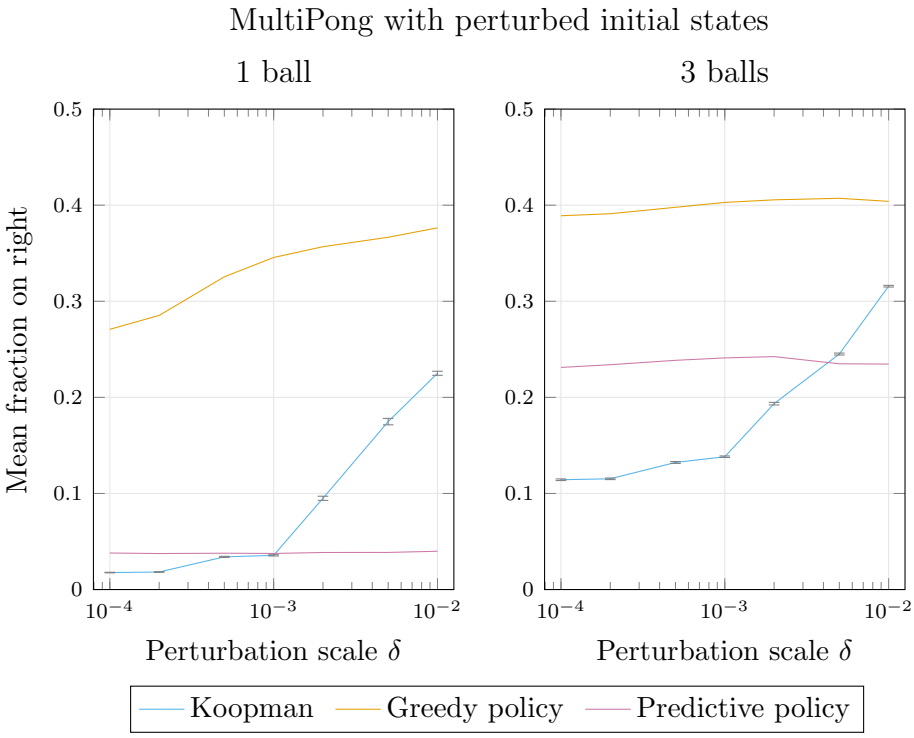

Figure 4.10: Performance of the non-robust Koopman control policy for the MultiPong system when the initial state is perturbed by an amount proportional to δ.

To combat this, we implement robust control policies with respect to the 1-norm initial state error. The results of these policies for the one-ball system for two values of δ are shown in Figure 4.11, for varying robustness bounds $\hat{\varepsilon}$. We see that the robust policies help us to achieve better performance for both cases, and that the larger δ perturbation benefits from a larger robustness bound.

We remark that, although EDMD is known to converge to the true Koopman operator [36], quantification of point-wise prediction errors as in (4.19) in the finite setting remains an open problem. Probabilistic bounds finite-data and infinite-data EDMD matrices as in [52] and [55] may serve as a starting point, but these do not account for errors due to the dictionary not being Koopman-invariant.

Nonetheless, even if true bounds could be obtained for $\varepsilon(u)$, the resulting control policy would likely prove to be overly conservative, unless there is reason to expect that

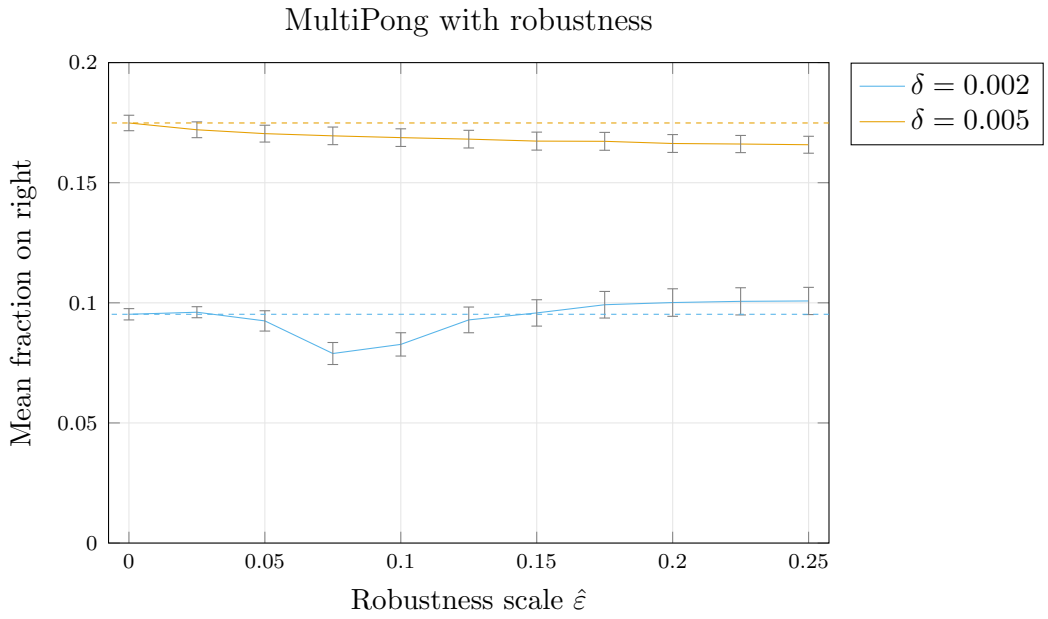

Figure 4.11: Performance of the robust Koopman control policy for the MultiPong system when the initial state is perturbed by an amount proportional to δ. Various values of robustness bound $\hat{\varepsilon}$ for the initial state 1-norm error are tested. The dashed lines mark the non-robust performance (i.e. $\hat{\varepsilon} = 0$).

the true error are closely aligned with the adversarial ones we consider. Instead, one may treat them as tunable design parameters as in Example 4.9.

Chapter 5

Directions for Future Research

The Koopman operator is a powerful tool that provides a linear description of nonlinear systems. With appropriate choice of observables, we can learn a finite-dimensional approximation that allows for practical application. We have shown how we may represent complex entity-based systems in the Koopman operator framework, and we demonstrated how we may design control policies that take advantage of the linearity. However, there remain a number of open questions to be explored by future work.

When including outputs in the Koopman representation, it may be interesting to consider how the role of the output differs from that of the state. For example, since output error does not propagate to future time steps, it may be that not all errors are equally bad. For control application in particular, identification of the optimal input relies only on the relative costs of different sequences. Thus, when learning the output approximation, it may be beneficial to explore approaches that better preserve this relative ordering, even if the absolute error is worse.

For representation of entity-based systems via the Koopman operator, we focused on density observables and their product, and found success for the systems we considered. However, it would be worth considering other types of observables, particularly since

product densities can yield large dictionaries. As a starting point, we may note that the entities in the systems we studied were only loosely coupled through sparse interactions. By quantifying this sparsity, we may consider how other types of observables may be able to exploit it to describe the system via a smaller model.

In terms of control, there are a number of directions that could be explored further. Our approach to control provides a piecewise linear form that is efficiently computable for small input sets and short horizons. The pruning algorithm we present addresses the horizon length, but may be poorly suited for large input sets as this would result in aggressive pruning at each time step. For inexact Koopman models, it is also unclear how much benefit we gain from extending the horizon. Although a longer horizon allows more flexibility in planning further ahead, the propagation of errors results in greater uncertainty as we look further into the future. This trade-off would be worth examining in future work.

Regarding the pruning algorithm, although it works well for the systems we considered, it is a heuristic method that does not provide guarantees. Application to other systems, or further study on quantifying the suboptimality of the pruned cost-to-go function would be useful in this regard, and may also provide insight on any modifications that could be made to improve the control performance. We may also consider reformulations that reduce the computational time, for example by enabling parallelization.

For our robust control policies, we considered error bounds as a design parameter, rather than attempting to identify a bound on the actual errors. It would be of interest to further study how these robustness parameters may be chosen, and in what ways they relate to empirical or analytical quantifications of the actual error. It may also be beneficial to study how whether a robust formulation could be derived for non-uniform error bounds, as this may allow us to defend against larger errors in the places where it matters without being overly conservative in others.

Bibliography

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

[2] D. Antunes and W. P. M. Heemels, "Linear quadratic regulation of switched systems using informed policies," *IEEE Transactions on Automatic Control*, vol. 62, no. 6, pp. 2675–2688, Jun. 2017.

[3] P. Arathoon and M. D. Kvalheim, "Koopman Embedding and Super-Linearization Counterexamples with Isolated Equilibria," Jul. 2023. arXiv: `2306.15126 [math]`.

[4] H. Arbabi and I. Mezić, "Ergodic Theory, Dynamic Mode Decomposition, and Computation of Spectral Properties of the Koopman Operator," *SIAM Journal on Applied Dynamical Systems*, vol. 16, pp. 2096–2126, Nov. 2017.

[5] C. Bakker, K. E. Nowak, and W. Steven Rosenthal, "Learning Koopman Operators for Systems with Isolated Critical Points," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, Dec. 2019, pp. 7733–7739.

[6] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013.

[7] R. Bellman, "Dynamic Programming." Princeton: University Press, 1957.

[8] A. Bemporad, A. Giua, and C. Seatzu, "An iterative algorithm for the optimal control of continuous-time switched linear systems," in *Sixth International Workshop on Discrete Event Systems, 2002. Proceedings.*, Oct. 2002, pp. 335–340.

[9] M. Blischke and J. P. Hespanha, "Learning Switched Koopman Models for Control of Entity-Based Systems," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, Dec. 2023, pp. 6006–6013.

[10] M. Blischke and J. P. Hespanha, "Optimal Control of Entity-Based Systems via Koopman Representations with Product Density Observables," *IEEE Open Journal of Control Systems*, pp. 1–15, 2026.

[11] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-Driven Control of Soft Robots Using Koopman Operator Theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, Jun. 2021.

[12] D. Bruder, X. Fu, and R. Vasudevan, "Advantages of bilinear Koopman realizations for the modeling and control of systems with unknown dynamics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4369–4376, Jul. 2021.

[13] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman Invariant Subspaces and Finite Linear Representations of Nonlinear Dynamical Systems for Control," *PLOS ONE*, vol. 11, no. 2, e0150171, Feb. 2016.

[14] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman theory for dynamical systems," *SIAM Review*, vol. 64, no. 2, pp. 229–340, May 2022.

[15] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, Dec. 2012.

[16] D. Cheng, Z. Xu, and T. Shen, "Equivalence-Based Model of Dimension-Varying Linear Systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 12, pp. 5444–5449, Dec. 2020.

[17] D. Deplano, N. Bastianello, M. Franceschelli, and K. H. Johansson, "Optimization and Learning in Open Multi-Agent Systems," *IEEE Transactions on Automatic Control*, pp. 1–16, 2026.

[18] J. F. Durán-Siguenza, L. I. Minchala, L. E. Garza-Castañón, and H. Zhang, "Control based on the Koopman operator: A comprehensive review," *Journal of the Franklin Institute*, vol. 362, no. 18, p. 108 256, Dec. 2025.

[19] C. Folkestad and J. W. Burdick, "Koopman NMPC: Koopman-based learning and nonlinear model predictive control of control-affine systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 7350–7356.

[20] C. Folkestad, Y. Chen, A. D. Ames, and J. W. Burdick, "Data-Driven Safety-Critical Control: Synthesizing Control Barrier Functions With Koopman Operators," *IEEE Control Systems Letters*, vol. 5, no. 6, pp. 2012–2017, Dec. 2021.

[21] R. Ghosh and M. Mcafee, "Koopman operator theory and dynamic mode decomposition in data-driven science and engineering: A comprehensive review," *Mathematical Modelling and Numerical Simulation with Applications*, vol. 4, no. 4, pp. 562–594, Dec. 2024.

[22] D. Goswami and D. A. Paley, "Global bilinearization and controllability of control-affine nonlinear systems: A Koopman spectral approach," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 6107–6112.

[23] D. M. Grobman, "Homeomorphism of systems of differential equations," *Doklady Akademii Nauk SSSR*, vol. 128, no. 5, pp. 880–881, 1959.

[24] D. A. Haggerty, M. J. Banks, E. Kamenar, A. B. Cao, P. C. Curtis, I. Mezić, and E. W. Hawkes, "Control of soft robots with inertial dynamics," *Science Robotics*, vol. 8, no. 81, eadd6864, Aug. 2023.

[25] P. C. Hansen, "Truncated Singular Value Decomposition Solutions to Discrete Ill-Posed Problems with Ill-Determined Numerical Rank," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, pp. 503–518, May 1990.

[26] P. Hartman, "A Lemma in the Theory of Structural Stability of Differential Equations," *Proceedings of the American Mathematical Society*, vol. 11, no. 4, pp. 610–620, 1960. JSTOR: 2034720.

[27] M. Haseli and J. Cortés, "Modeling nonlinear control systems via Koopman Control Family: Universal forms and subspace invariance proximity," *Automatica*, vol. 185, Mar. 2026.

[28] M. Haseli, I. Mezić, and J. Cortés, "Two Roads to Koopman Operator Theory for Control: Infinite Input Sequences and Operator Families," Oct. 2025. arXiv: 2510.15166 [math].

[29] A. Hasnain, N. Boddupalli, S. Balakrishnan, and E. Yeung, "Steady state programming of controlled nonlinear systems via deep dynamic mode decomposition," in *2020 American Control Conference (ACC)*, Jul. 2020, pp. 4245–4251.

[30] M. Hemati and H. Yao, "Dynamic Mode Shaping for Fluid Flow Control: New Strategies for Transient Growth Suppression," in *8th AIAA Theoretical Fluid Mechanics Conference*, Denver, Colorado: American Institute of Aeronautics and Astronautics, Jun. 2017.

[31] B. Huang, X. Ma, and U. Vaidya, "Feedback Stabilization Using Koopman Operator," in *2018 IEEE Conference on Decision and Control (CDC)*, Dec. 2018, pp. 6434–6439.

[32] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of Koopman eigenfunctions for control," *Machine Learning: Science and Technology*, vol. 2, no. 3, p. 035 023, Jun. 2021.

[33] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, May 1931.

[34] B. O. Koopman and J. v. Neumann, "Dynamical Systems of Continuous Spectra," *Proceedings of the National Academy of Sciences*, vol. 18, no. 3, pp. 255–263, Mar. 1932.

[35] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, Jul. 2018.

[36] M. Korda and I. Mezić, "On convergence of extended dynamic mode decomposition to the Koopman operator," *Journal of Nonlinear Science*, vol. 28, no. 2, pp. 687–710, Apr. 2018.

[37] M. Korda, Y. Susuki, and I. Mezić, "Power grid transient stabilization using Koopman model predictive control," *IFAC-PapersOnLine*, 10th IFAC Symposium on Control of Power and Energy Systems CPES 2018, vol. 51, no. 28, pp. 297–302, Jan. 2018.

[38] Y. Lan and I. Mezić, "Linearization in the large of nonlinear systems and Koopman operator spectrum," *Physica D: Nonlinear Phenomena*, vol. 242, no. 1, pp. 42–53, Jan. 2013.

[39] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, "Learning Compositional Koopman Operators for Model-Based Control," in *Eighth International Conference on Learning Representations*, Apr. 2020.

[40] E. Ling, L. Ratliff, and S. Coogan, "Koopman Operator Approach for Instability Detection and Mitigation in Signalized Traffic," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 1297–1302.

[41] Z. Liu, N. Ozay, and E. D. Sontag, "Properties of immersions for systems with multiple limit sets with implications to learning Koopman embeddings," *Automatica*, vol. 176, p. 112 226, Jun. 2025.

[42] W. A. Manzoor, S. Rawashdeh, and A. Mohammadi, "Vehicular Applications of Koopman Operator Theory—A Survey," *IEEE Access*, vol. 11, pp. 25 917–25 931, 2023.

[43] A. Mauroy, I. Mezić, and Y. Susuki, Eds., "The Koopman Operator in Systems and Control" (Lecture Notes in Control and Information Sciences 484), 1st. Springer, 2020.

[44] I. Mezić, "Spectral Properties of Dynamical Systems, Model Reduction and Decompositions," *Nonlinear Dynamics*, vol. 41, no. 1, pp. 309–325, Aug. 2005.

[45] I. Mezić, "Spectrum of the Koopman Operator, Spectral Expansions in Functional Spaces, and State-Space Geometry," *Journal of Nonlinear Science*, vol. 30, no. 5, pp. 2091–2145, Oct. 2020.

[46] I. Mezić, "Koopman operator, geometry, and learning of dynamical systems," *Notices of the American Mathematical Society*, vol. 68, no. 07, p. 1, Aug. 2021.

[47] I. Mezić and A. Banaszuk, "Comparison of systems with complex behavior," *Physica D: Nonlinear Phenomena*, vol. 197, no. 1, pp. 101–133, Oct. 2004.

[48] A. Mironchenko, "Live Systems of Varying Dimension: Modeling and Stability," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, Dec. 2023, pp. 3956–3961.

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[50] Y. Naddaf, "Game-independent AI agents for playing Atari 2600 console games," M.S. thesis, University of Alberta, 2010.

[51] A. Narasingam and J. S.-I. Kwon, "Koopman Lyapunov-based model predictive control of nonlinear chemical process systems," *AIChE Journal*, vol. 65, no. 11, e16743, 2019.

[52] F. Nüske, S. Peitz, F. Philipp, M. Schaller, and K. Worthmann, "Finite-Data Error Bounds for Koopman-Based Prediction and Control," *Journal of Nonlinear Science*, vol. 33, no. 1, p. 14, Feb. 2023.

[53] S. E. Otto and C. W. Rowley, "Koopman operators for estimation and control of dynamical systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 59–87, Feb. 2021.

[54] S. Peitz and S. Klus, "Koopman operator-based model reduction for switched-system control of PDEs," *Automatica*, vol. 106, pp. 184–191, Aug. 2019.

[55] F. M. Philipp, M. Schaller, S. Boshoff, S. Peitz, F. Nüske, and K. Worthmann, "Variance representations and convergence rates for data-driven approximations of Koopman operators," May 2024. arXiv: `2402.02494 [math]`.

[56] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, Jan. 2016.

[57] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Generalizing Koopman theory to allow for inputs and control," *SIAM Journal on Applied Dynamical Systems*, vol. 17, no. 1, pp. 909–930, Jan. 2018.

[58] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson, "Spectral analysis of nonlinear flows," *Journal of Fluid Mechanics*, vol. 641, pp. 115–127, Dec. 2009.

[59] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of Fluid Mechanics*, vol. 656, pp. 5–28, Aug. 2010.

[60] L. Shi, M. Haseli, G. Mamakoukas, D. Bruder, I. Abraham, T. Murphey, J. Cortés, and K. Karydis, "Koopman Operators in Robot Learning," *IEEE Transactions on Robotics*, vol. 42, pp. 1088–1107, 2026.

[61] R. Strässer, K. Worthmann, I. Mezić, J. Berberich, M. Schaller, and F. Allgöwer, "An overview of Koopman-based control: From error bounds to closed-loop guarantees," *Annual Reviews in Control*, vol. 61, p. 101 035, Jan. 2026.

[62] A. Surana, "Koopman operator based observer synthesis for control-affine nonlinear systems," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec. 2016, pp. 6492–6499.

[63] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: Theory and applications," *Journal of Computational Dynamics*, vol. 1, no. 2, pp. 391–421, Dec. 2014.

[64] M. O. Williams, M. S. Hemati, S. T. M. Dawson, I. G. Kevrekidis, and C. W. Rowley, "Extending data-driven Koopman analysis to actuated systems," *IFAC-PapersOnLine*, 10th IFAC Symposium on Nonlinear Control Systems NOLCOS 2016, vol. 49, no. 18, pp. 704–709, Jan. 2016.

[65] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, Dec. 2015.

[66] Z. Wu and Q. He, "Optimal switching sequence for switched linear systems," *SIAM Journal on Control and Optimization*, vol. 58, no. 2, pp. 1183–1206, Jan. 2020.

[67] W. Xu, Z. G. Feng, G.-H. Lin, and L. Yu, "Optimal scheduling of discrete-time switched linear systems," *IMA Journal of Mathematical Control and Information*, vol. 37, no. 1, pp. 1–18, Mar. 2020.

[68] W. Xu, Z. G. Feng, J. W. Peng, and K. F. C. Yiu, "Optimal switching for linear quadratic problem of switched systems in discrete time," *Automatica*, vol. 78, pp. 185–193, Apr. 2017.

[69] W. Zhang, J. Hu, and A. Abate, "On the value functions of the discrete-time switched LQR problem," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2669–2674, Nov. 2009.

[70] X. Zhang, W. Pan, R. Scattolini, S. Yu, and X. Xu, "Robust tube-based model predictive control with Koopman operators," *Automatica*, vol. 137, p. 110 114, Mar. 2022.

[71] J. Zhao, M. Gan, and G. Chen, "Optimal control of discrete-time switched linear systems," *Journal of the Franklin Institute*, vol. 357, no. 9, pp. 5340–5358, Jun. 2020.

[72] F. Zhu and P. J. Antsaklis, "Optimal control of hybrid switched systems: A brief survey," *Discrete Event Dynamic Systems*, vol. 25, no. 3, pp. 345–364, Sep. 2015.