; revised ; date of current version 10 November 2025.

Optimal Control of Entity-Based Systems via Koopman Representations with Product Density Observables

MADELINE BLISCHKE[1] **AND JOÃO P. HESPANHA**[1]

[1]University of California, Santa Barbara, CA 93106 USA

CORRESPONDING AUTHOR: Madeline Blischke (e-mail: blischke@ucsb.edu)

This material is based upon work supported by the U.S. Office of Naval Research grant number N00014-23-1-2708 and by the National Science Foundation grant number 2229876.

ABSTRACT 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. This work considers a nonlinear optimal control problem with final state constraints, which we alternatively represent by a linear problem on a lifted state space described by the Koopman operator. 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 provide a heuristic pruning algorithm that reduces the search space to a much smaller subset. Our control approach is generally applicable, but its effectiveness depends on a domain-specific choice of lifting functions, termed *observables*. For a class of systems with dynamics described by some number of distinct "entities", we define observables that describe "densities" of these entities, as well as products of densities that capture a richer class of interactions between entities.

INDEX TERMS Koopman operator, data-driven modeling, optimal control

I. INTRODUCTION

Whereas many well-studied and general techniques exist for the analysis and control of linear dynamical systems, there are comparatively few that exist for nonlinear systems. However, techniques for linear systems can be transferred to the nonlinear setting if one can identify an approximate linearization of nonlinear dynamics. One approach is via the Koopman operator, first introduced in [1], which departs from the traditional state-based description of dynamical systems in favor of an operator-theoretic description. By considering the evolution of functions of the state, nonlinear state dynamics are described by a linear operator on a function space. Moreover, this linear operator captures global system behavior, in contrast to, e.g., a Jacobian-based linearization that is generally valid only within a local neighborhood of an equilibrium point.

Although the Koopman operator was first introduced nearly a hundred years ago [1], it has received the greatest attention in the past few decades, see e.g. [2]–[4]. Study of Koopman operator theory has advanced existing linearization results, such as the extension of the Hartman-Grobman theorem to the full basin of attraction [5]. Koopman-based representations can even describe systems with multiple isolated equilibria [6], though with limitations as discussed in [7]. Applications of the Koopman operator are commonly facilitated by data-driven approximations such as via extended dynamic mode decomposition [8], which is known to possess desirable convergence properties [9].

Most relevant to this work is the extensive literature on applying Koopman-based methods to control, which is the focus of the survey [10]. Linear predictors such as considered in [11] are immediately amenable to a variety of control techniques, and have seen successful application to a variety

of problems [12], [13]. The use of more general bilinear forms has also seen success, particularly for systems with control-affine dynamics [14], [15]. A contrasting approach is the use of an input-parametrized Koopman model, described by [16] as the "Koopman Control Family". This form naturally describes systems with finite sets of inputs [17], [18], but has also seen application to other systems via a discretization of a continuous input space [19].

This paper builds upon and extends the framework introduced in [17]. We consider an optimal control problem for a nonlinear dynamical system, for which we wish to minimize cumulative cost over a finite horizon subject to final state constraints. Leveraging the Koopman operator, we obtain a linear approximation of the dynamics on a lifted state space, and similarly use linear approximations of the nonlinear cost and constraint functions. This leads to a piecewise linear optimal cost-to-go, for which we derive a compact representation via an optimization problem with a linear objective subject to linear state constraints.

If we can obtain an exact Koopman representation, then the optimization problem we derive is equivalent to the original nonlinear problem. However, in general we obtain only an inexact approximation of the Koopman operator, and so there will be some amount of mismatch. In light of this, we consider a robust version of the optimal control problem in which we minimize the worst-case cost with respect to small, bounded errors in predictions from the Koopman model and in the initial state measurement. We show that, perhaps surprisingly, the problem admits a piecewise affine cost-to-go, similar to the nominal problem.

Although the linear Koopman representation provides a more tractable alternative to the nonlinear control problem, the combinatorial nature of the dynamics remains a challenge, as the number of potential input sequences one must search over grows exponentially in the horizon length. As such, we consider a suboptimal approach that searches over a much smaller subset of sequences. We present a dynamic pruning algorithm that recursively constructs such a subset via a heuristic inspired by the concepts of "exploitation" and "exploration" commonly considered in reinforcement learning.

Whereas the control approach we present is generally applicable to a broad range of systems, another focus of this paper is specific to a class "entity-based systems" as described in [17]. These are systems for which the state and dynamics are described by those of a number of distinct "entities", such as agents in strategy games, vehicles in traffic control problems, or individuals in a dynamic population. One notable example of such systems is found in the domain of arcade games, which present a wide range of complex combinatorial problems. Games for the Atari 2600 console are of particular note as a common benchmark for reinforcement learning, such as in work by Google DeepMind on deep Q-networks [20], and are freely available via a programming interface designed specifically for reinforcement learning [21].

In application of the Koopman operator, the choice of appropriate lifting functions used in learning Koopman representation is often a challenging domain-specific tasks. For representation of entity-based systems, this paper further develops the type of observables introduced in [17], which capture a notion of the "density" of entities in specific regions of their state space. We discuss some limitations of these densities, including a demonstration of a system for which they are insufficient. By augmenting the set of lifting functions with so-called "product densities" between different entity classes, we show that we are able to accurately capture a richer variety of dynamics.

The remainder of this paper is structured as follows. Section II reviews relevant properties of the Koopman operator. Section III provides results on optimal control using Koopman approximations with state constraints and robustness. Section IV defines the notion of an entity-based system and introduces the observables we use for modeling them. Sections V provides numerical results supporting our methods. Section VI provides concluding remarks.

II. THE KOOPMAN OPERATOR

Consider the discrete-time dynamical system

$$x^+ = f(x,u), \qquad y = g(x) \tag{1}$$

with state $x \in \mathcal{X}$, output $y \in \mathbb{R}^{n_y}$, and input $u \in \mathcal{U}$. The notation x^+ denotes the state at the next time step. For now, we do not impose a specific numerical representation on the state x. We focus on systems for which the input set $\mathcal{U}$ is a finite set with size n_u, but one could also consider continuous input spaces via finite discretizations as in [19].

Whereas (1) describes the dynamics of the state, the associated Koopman operator instead describes the dynamics of functions of the state, termed *observables*. Formally, we define an input-dependent Koopman operator $\mathcal{K}_u$, acting on some appropriate function space $\mathcal{F}$ of observables $\phi : \mathcal{X} \to \mathbb{R}$, as

$$\mathcal{K}_u\phi(x) := \phi(f(x,u)), \qquad \phi \in \mathcal{F}. \tag{2}$$

That is, the Koopman operator $\mathcal{K}_u$ maps an observable ϕ to a new observable that evaluates the value of ϕ one time step ahead when the input u is applied. We say that a subspace $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ is *Koopman-invariant* if

$$\mathcal{K}_u\phi \in \tilde{\mathcal{F}}, \quad \forall \phi \in \tilde{\mathcal{F}},\ \forall u \in \mathcal{U}. \tag{3}$$

We summarize here the key properties of the Koopman operator used in this paper. The Koopman operator is always a linear operator on $\mathcal{F}$, even if the dynamics of the underlying system are nonlinear, since

$$\begin{aligned}\mathcal{K}_u[\alpha_1\phi_1(x) + \alpha_2\phi_2(x)] &= \alpha_1\phi_1(f(x,u)) + \alpha_2\phi_2(f(x,u)) \\ &= \alpha_1\mathcal{K}_u\phi_1(x) + \alpha_2\mathcal{K}_u\phi_2(x).\end{aligned}$$

From repeated application of (2), we see that compositions of the Koopman operator yield

$$\phi(x_k) = \mathcal{K}_{u_k}\mathcal{K}_{u_{k-1}}\cdots\mathcal{K}_{u_1}\phi(x_0), \tag{4}$$

where x_k is the state of (1) at time k resulting from applying the input sequence $(u_1, \dots, u_k) \in \mathcal{U}^k$ to the initial state x_0.

A. FINITE-DIMENSIONAL DICTIONARIES

In practice, we consider the action of the Koopman operator on a finite-dimensional subspace spanned by a chosen *dictionary* of observables $\mathbf{\Psi} := [\psi_1 \cdots \psi_N]^T$, for which the Koopman operator applies elementwise. The vector $\mathbf{\Psi}(x)$ is commonly referred to as the *lifted state*. We say that the dictionary $\mathbf{\Psi}$ is Koopman-invariant if the span of its entries

$$\operatorname{span} \mathbf{\Psi} := \left\{ \phi \in \mathcal{F} : \exists \alpha_1, \dots, \alpha_N \in \mathbb{R}, \text{ s.t. } \phi = \sum_{i=1}^{N} \alpha_i \psi_i \right\}$$

is a Koopman-invariant subspace as defined by (3).

Given a Koopman-invariant dictionary $\mathbf{\Psi}$, the linearity of the operators $\mathcal{K}_u$ guarantees the existence of matrices $A_u \in \mathbb{R}^{N \times N}$ such that

$$\mathcal{K}_u \mathbf{\Psi}(x) = A_u \mathbf{\Psi}(x). \tag{5}$$

In addition, if each entry of the output map g is in $\operatorname{span} \mathbf{\Psi}$, then there exists a matrix $C \in \mathbb{R}^{n_y \times N}$ such that

$$g(x) = C\mathbf{\Psi}(x). \tag{6}$$

Combining (4)-(6) allows for the prediction of the output k steps ahead as

$$y_k = g(x_k) = C A_{u_k} A_{u_{k-1}} \cdots A_{u_1} \mathbf{\Psi}(x_0). \tag{7}$$

It is easy to identify dictionaries that span a Koopman-invariant subspace (e.g. $\mathbf{\Psi}(x)$ equal to a constant), and dictionaries that contain the output map g in their span (e.g. $\mathbf{\Psi}(x) = g(x)$). However, it is generally difficult to identify a dictionary that satisfies both properties. As such, we typically work with dictionaries for which (5)-(7) do not hold exactly, but instead hold approximately with some error that we hope to make small.

B. DATA-DRIVEN APPROXIMATION

The matrices A_u and C used in (5) and (6) can be constructed from data via an algorithm called *extended dynamic mode decomposition* (EDMD) [8]. The algorithm requires a set of data snapshots

$$\{x_i, \hat{x}_i, u_i, y_i\}_{i=1}^{M} \tag{8}$$

satisfying

$$\hat{x}_i = f(x_i, u), \qquad y_i = g(x_i),$$

and some dictionary $\mathbf{\Psi}$. The ordering of the snapshots is irrelevant, and they may be taken from one long trajectory or several short ones.

We partition the snapshots by each input $u \in \mathcal{U}$ into input-specific sets $\{x_i, \hat{x}_i, y_i, u_i\}_{i \in \mathcal{I}(u)}$ where $\mathcal{I}(u) = \{i : u_i = u\}$. Each partition is then a set of snapshots for an autonomous system with a fixed input. For each input u, performing EDMD on the corresponding snapshots yields a matrix A_u as the solution to

$$A_u = \operatorname*{arg\,min}_{A \in \mathbb{R}^{N \times N}} \sum_{i \in \mathcal{I}(u)} \|A\mathbf{\Psi}(x_i) - \mathbf{\Psi}(\hat{x}_i)\|_2^2. \tag{9}$$

This solution can be computed as

$$A_u = H_u G_u^{\dagger},$$

where

$$G_u = \sum_{i \in \mathcal{I}(u)} \frac{1}{|\mathcal{I}(u)|} \mathbf{\Psi}(x_i)\mathbf{\Psi}(x_i)^T,$$

$$H_u = \sum_{i \in \mathcal{I}(u)} \frac{1}{|\mathcal{I}(u)|} \mathbf{\Psi}(\hat{x}_i)\mathbf{\Psi}(x_i)^T,$$

and $G_u^{\dagger}$ denotes the Moore-Penrose pseudoinverse. The matrix C is similarly obtained as the solution to

$$C = \operatorname*{arg\,min}_{C \in \mathbb{R}^{n_y \times N}} \sum_{i=1}^{M} \|C\mathbf{\Psi}(x_i) - y_i\|_2^2. \tag{10}$$

If $\mathbf{\Psi}$ is a Koopman-invariant dictionary whose span contains each element of g, then the optimal values to (9) and (10) are zero and (5) and (6) hold exactly. Otherwise, the optimal values are some positive numbers and (5) and (6) hold only approximately.

III. OPTIMAL CONTROL PROBLEM

Given an initial state x_0, we consider the optimal control problem

$$\min_{\boldsymbol{\mu} \in \mathcal{U}^h} \quad \sum_{k=0}^{h} c_k(x_k) \tag{11a}$$

$$\text{s.t.} \quad x_k = f(x_{k-1}, u_k), \quad k = 1, \dots, h, \tag{11b}$$

$$\boldsymbol{\zeta}(x_h) \le \mathbf{z}, \tag{11c}$$

where $c_k : \mathcal{X} \to \mathbb{R}$ for $k \in \{0, \dots, h\}$ are time-varying functions that accumulate cost at each time step, and the function $\boldsymbol{\zeta} : \mathcal{X} \to \mathbb{R}^{n_c}$ expresses a constraint on the final state that must be upper bounded elementwise by $\mathbf{z} \in \mathbb{R}^{n_c}$. If each c_k and each entry of $\boldsymbol{\zeta}$ are included as outputs in g, then by defining the vectors $\mathbf{c}_k \in \mathbb{R}^N$ and the matrix $Z \in \mathbb{R}^{n_c \times N}$ as the rows of the matrix C in (10) corresponding those outputs, we can approximate

$$c_k(x) \approx \mathbf{c}_k^T \mathbf{\Psi}(x), \qquad \boldsymbol{\zeta}(x) \approx Z\mathbf{\Psi}(x). \tag{12}$$

Combining this with (5), solving the problem (11) is equivalent to finding an input sequence $\boldsymbol{\mu} \in \mathcal{U}^h$ that minimizes $J_0(x_0)$, where J_k is the optimal cost-to-go defined by

$$J_k(x_k) := \min_{\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}} \quad \sum_{\ell=k}^{h} \mathbf{c}_\ell^T \boldsymbol{\xi}_\ell \tag{13a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_k = \mathbf{\Psi}(x_k) \tag{13b}$$

$$\boldsymbol{\xi}_\ell = A_{u_\ell} \boldsymbol{\xi}_{\ell-1}, \quad \ell \in \{k+1, \dots, h\}, \tag{13c}$$

$$Z\boldsymbol{\xi}_h \le \mathbf{z}. \tag{13d}$$

Here $\boldsymbol{\mu}_k^h := (u_{k+1}, \dots, u_h)$ denotes the sequence of controls applied from time k through time h. The case $k = h$ represents the empty sequence, which we denote by $\boldsymbol{\mu}_h^h = \emptyset$.

Note that since (5) and (12) are unlikely to hold exactly, there will likely be a gap between the costs of (11) and (13).

It is also possible that some input sequences that satisfy the constraints for (13) may not satisfy the constraints for (11) and vice versa. We shall first consider the solution to (13) as written, and then we shall present a robust control law that accounts for such errors.

Theorem 1:
The optimal cost-to-go J_k is piecewise linear in the lifted state $\boldsymbol{\Psi}(x_k)$, and is equal to the solution of the constrained optimization problem

$$J_k(x_k) = \min_{\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}} \quad \mathbf{d}(\boldsymbol{\mu}_k^h)^T \boldsymbol{\Psi}(x_k) \tag{14a}$$

$$\text{s.t.} \quad W(\boldsymbol{\mu}_k^h)\boldsymbol{\Psi}(x_k) \leq \mathbf{z}, \tag{14b}$$

where, for each $\boldsymbol{\mu}_k^h = (u_{k+1}, \dots, u_h)$, we define

$$\mathbf{d}(\boldsymbol{\mu}_k^h) := \sum_{\ell=k}^{h} A_{u_{k+1}}^T \cdots A_{u_\ell}^T \mathbf{c}_\ell, \tag{15}$$

$$W(\boldsymbol{\mu}_k^h) := Z A_{u_h} \cdots A_{u_{k+1}}. \tag{16}$$

The optimal input sequence is obtained as the argmin of (14).

Proof:
For a given sequence $\boldsymbol{\mu}_k^h$, recursive application of (13c) gives

$$\boldsymbol{\xi}_\ell = A_{u_\ell} \cdots A_{u_{k+1}} \boldsymbol{\xi}_k, \quad \forall \ell \in \{k, \dots, h\}.$$

Using (15)-(16), we rewrite the cost (13a) and the left side of the inequality (13d) as

$$\sum_{\ell=k}^{h} \mathbf{c}_\ell^T \boldsymbol{\xi}_\ell = \sum_{\ell=k}^{h} \mathbf{c}_\ell^T A_{u_\ell} \cdots A_{u_{k+1}} \boldsymbol{\xi}_k = \mathbf{d}(\boldsymbol{\mu}_k^h)^T \boldsymbol{\xi}_k,$$

$$Z\boldsymbol{\xi}_h = Z A_{u_\ell} \cdots A_{u_{k+1}} \boldsymbol{\xi}_k = W(\boldsymbol{\mu}_k^h)\boldsymbol{\xi}_k,$$

which gives us (14) as desired. That the optimal input sequence is obtained as the argmin follows directly from the definition of the cost-to-go in (13). ∎

We further note that the $\mathbf{d}(\boldsymbol{\mu}_k^h)$ vectors and $W(\boldsymbol{\mu}_k^h)$ matrices can be constructed recursively by letting $\mathbf{d}(\emptyset) = \mathbf{c}_h$, $W(\emptyset) = Z$, and computing

$$\mathbf{d}(\boldsymbol{\mu}_{k-1}^h) = A_{u_k}^T \mathbf{d}(\boldsymbol{\mu}_k^h) + \mathbf{c}_{k-1}, \quad k \in \{h, \dots, 1\}, \tag{17}$$

$$W(\boldsymbol{\mu}_{k-1}^h) = W(\boldsymbol{\mu}_k^h) A_{u_k}, \quad k \in \{h, \dots, 1\}. \tag{18}$$

A. INCORPORATING ROBUSTNESS

There are multiple potential sources of error that may be helpful to guard against. We consider here two types of error. The first is error in the initial lifted state, typically arising from measurement noise, which results in a mismatch between the actual system state and what the controller believes it to be. The second is prediction error due to the non-exact version of (5). Suppose that, for some $p \in [1, \infty]$, the p-norm of the initial state error is bounded by some $\hat{\varepsilon} > 0$, and that the p-norm of the prediction errors of $x^+ \approx A_u x$ are bounded by some input-dependent $\varepsilon_u > 0$. When an input sequence $\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}$ is applied from the state $x_k \in \mathcal{X}$, the set of all possible lifted state trajectories that satisfy these bounds is given by

$$\begin{aligned} \mathcal{T}(x_k, \boldsymbol{\mu}_k^h) := \Big\{ (\boldsymbol{\xi}_k, \dots, \boldsymbol{\xi}_h) \in \left(\mathbb{R}^N\right)^{h-k+1} \; : \\ \|\boldsymbol{\xi}_k - \hat{\boldsymbol{\Psi}}(x_k)\|_p \leq \hat{\varepsilon}, \\ \|\boldsymbol{\xi}_\ell - A_{u_\ell} \boldsymbol{\xi}_{\ell-1}\|_p \leq \varepsilon_{u_\ell} \;\; \forall \ell \in \{k+1, \dots, h\} \Big\}, \end{aligned} \tag{19}$$

where $\hat{\boldsymbol{\Psi}}(x_k)$ denotes our (possibly inaccurate) estimate of $\boldsymbol{\Psi}(x_k)$. We then wish to choose an input sequence $\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}$ that minimizes the worst-case cost among trajectories in this set, while ensuring that all trajectories satisfy the final state constraint. This is expressed by the robust cost-to-go J_k^ε defined by

$$J_k^\varepsilon(x_k) := \min_{\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}} \; \max_{(\boldsymbol{\xi}_k, \dots, \boldsymbol{\xi}_h) \in \mathcal{T}(x_k, \boldsymbol{\mu}_k^h)} \sum_{\ell=k}^{h} \mathbf{c}_\ell^T \boldsymbol{\xi}_\ell \tag{20a}$$

$$\text{s.t. } Z\boldsymbol{\xi}_h \leq \mathbf{z}, \;\; \forall (\boldsymbol{\xi}_k, \dots, \boldsymbol{\xi}_h) \in \mathcal{T}(x_k, \boldsymbol{\mu}_k^h). \tag{20b}$$

It turns out that the solution to this robust optimization also has a constrained piecewise affine structure, as described in the following result.

Theorem 2:
The robust cost-to-go J_k^ε is piecewise affine with respect to the estimate $\hat{\boldsymbol{\Psi}}(x_k)$, and is equal to the solution of the following constrained optimization problem

$$J_k^\varepsilon(x_k) = \min_{\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}} \quad \mathbf{d}(\boldsymbol{\mu}_k^h)^T \hat{\boldsymbol{\Psi}}(x_k) + \beta(\boldsymbol{\mu}_k^h), \tag{21a}$$

$$\text{s.t.} \quad W(\boldsymbol{\mu}_k^h)\hat{\boldsymbol{\Psi}}(x_k) \leq \boldsymbol{\omega}(\boldsymbol{\mu}_k^h), \tag{21b}$$

with the $\mathbf{d}(\boldsymbol{\mu}_k^h)$ and $W(\boldsymbol{\mu}_k^h)$ defined in (15) and (16), and $\beta(\boldsymbol{\mu}_k^h)$ and $\boldsymbol{\omega}(\boldsymbol{\mu}_k^h)$ defined by

$$\beta(\boldsymbol{\mu}_k^h) := \hat{\varepsilon}\|\mathbf{d}(\boldsymbol{\mu}_k^h)\|_q + \sum_{\ell=k+1}^{h} \varepsilon_{u_\ell} \|\mathbf{d}(\boldsymbol{\mu}_\ell^h)\|_q,$$

$$\boldsymbol{\omega}(\boldsymbol{\mu}_k^h) := \mathbf{z} - \hat{\varepsilon}\boldsymbol{\rho}_q\left(W(\boldsymbol{\mu}_k^h)\right) - \sum_{\ell=k+1}^{h} \varepsilon_{u_\ell} \boldsymbol{\rho}_q\left(W(\boldsymbol{\mu}_\ell^h)\right),$$

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

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

Proof:
Note that the trajectory $(\boldsymbol{\xi}_k, \dots, \boldsymbol{\xi}_h)$ belongs to the set $\mathcal{T}(x_k, \boldsymbol{\mu}_k^h)$ defined by (19) if and only if there exists vectors $\mathbf{v}_k, \dots, \mathbf{v}_h \in \mathbb{R}^N$ with $\|\mathbf{v}_k\|_p \leq \hat{\varepsilon}$ and $\|\mathbf{v}_\ell\|_p \leq \varepsilon_{u_\ell}, \ell \in \{k+1, \dots, h\}$ such that

$$\boldsymbol{\xi}_k = \hat{\boldsymbol{\Psi}}(x_k) + \mathbf{v}_k,$$

$$\boldsymbol{\xi}_\ell = A_{u_\ell} \boldsymbol{\xi}_{\ell-1} + \mathbf{v}_\ell, \quad \forall \ell \in \{k+1, \dots, h\}.$$

Recursively iterating these expressions, we then have

$$\boldsymbol{\xi}_m = A_{u_m} \cdots A_{u_{k+1}} \hat{\boldsymbol{\Psi}}(x_k) + \sum_{\ell=k}^{m} A_{u_m} \cdots A_{u_{\ell+1}} \mathbf{v}_\ell. \quad (22)$$

Using the specific expression for the case $\ell = h$, the vector inequality in (20b) becomes

$$Z\boldsymbol{\xi}_h = W(\boldsymbol{\mu}_k^h)\hat{\boldsymbol{\Psi}}(x_k) + \sum_{\ell=k}^{h} W(\boldsymbol{\mu}_\ell^h)\mathbf{v}_\ell \leq \mathbf{z},$$

which must be satisfied for all $\mathbf{v}_k, \ldots, \mathbf{v}_h \in \mathbb{R}^N$ such that $\|\mathbf{v}_k\|_p \leq \hat{\varepsilon}$ and $\|\mathbf{v}_\ell\|_p \leq \varepsilon_{u_\ell}$, $\ell \in \{k+1, \ldots, h\}$. Thus, for each $i \in \{1, \ldots, n_c\}$ we must enforce that

$$\mathbf{w}_i(\boldsymbol{\mu}_k^h)^T \hat{\boldsymbol{\Psi}}(x_k) + \max_{\substack{\mathbf{v}_k, \ldots, \mathbf{v}_h \in \mathbb{R}^N \\ \text{s.t. } \|\mathbf{v}_k\|_p \leq \hat{\varepsilon} \\ \|\mathbf{v}_\ell\|_p \leq \varepsilon_{u_\ell},\, \ell \in \{k+1, \ldots, h\}}} \sum_{\ell=k}^{h} \mathbf{w}_i(\boldsymbol{\mu}_\ell^h)^T \mathbf{v}_\ell \leq z_i, \quad (23)$$

where $\mathbf{w}_i(\boldsymbol{\mu}_k^h)$ is the ith row of $W(\boldsymbol{\mu}_k^h)$, and z_i is the ith entry of $\mathbf{z}$. Due to decoupling among the $\mathbf{v}_\ell$, the maximization above is achieved by maximizing each term individually. For any $p, q \in [1, \infty]$ with $\frac{1}{p} + \frac{1}{q} = 1$, the norms $\|\cdot\|_p$ and $\|\cdot\|_q$ are dual to each other, and so for any vector $\mathbf{w}$ and $\varepsilon > 0$, we have

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

Thus, (23) becomes

$$\mathbf{w}_i(\boldsymbol{\mu}_k^h)^T \hat{\boldsymbol{\Psi}}(x_k) + \hat{\varepsilon}\|\mathbf{w}_i(\boldsymbol{\mu}_k^h)\|_q + \sum_{\ell=k+1}^{h} \varepsilon_{u_\ell} \|\mathbf{w}_i(\boldsymbol{\mu}_\ell^h)\|_q \leq z_i,$$

and now subtracting the summation from both sides yields the ith constraint of (21b).

By a similar argument, substituting (22) into (20a) gives

$$\sum_{m=k}^{h} \mathbf{c}_m^T \boldsymbol{\xi}_m = \mathbf{d}(\boldsymbol{\mu}_k^h)^T \hat{\boldsymbol{\Psi}}(x_k) + \sum_{\ell=k}^{h} \mathbf{d}(\boldsymbol{\mu}_\ell^h)^T \mathbf{v}_\ell,$$

which yields the objective (21a) when maximized subject to $\|\mathbf{v}_k\|_p \leq \hat{\varepsilon}$ and $\|\mathbf{v}_\ell\|_p \leq \varepsilon_{u_\ell}, \ell \in \{k+1, \ldots, h\}$. This completes the proof. ■

B. PRUNING ALGORITHM

For small horizons h, it is feasible to perform the constrained optimizations in (14) and (21) over all input sequences $\boldsymbol{\mu}_k^h \in \mathcal{U}^{k-h}$. However, as the number of sequences grows exponentially in h, this approach becomes intractable for large horizons. We propose here a heuristic algorithm that uses the concepts of "exploration" and "exploitation" from reinforcement learning to construct a subset of inputs that may not contain the optimal sequence for all states, but that hopefully contains sequences that achieve costs close to the optimal.

We present the algorithm for the robust optimization in (21). The optimization in (14) is equivalent to the special case with $\hat{\varepsilon} = 0$ and $\varepsilon_u = 0, \forall u \in \mathcal{U}$, as we would then have $\beta(\boldsymbol{\mu}) = 0$ and $\boldsymbol{\omega}(\boldsymbol{\mu}) = \mathbf{z}$ for all $\boldsymbol{\mu} \in \mathcal{U}^h$.

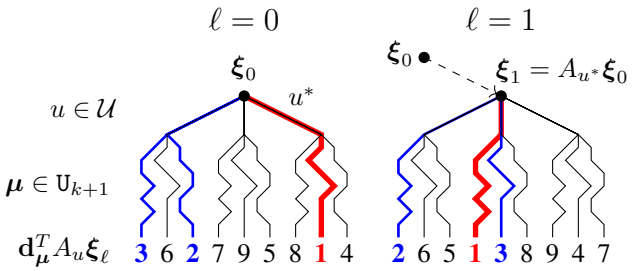

FIGURE 1. An illustrated example of two iterations in Algorithm 2. From an initial state $\boldsymbol{\xi}_0$, the costs achieved by each sequence $u\boldsymbol{\mu} \in \mathcal{U} \otimes \mathtt{U}_{k+1}$ is computed. The best such sequence is highlighted in red, while the next best sequences are highlighted in blue. The red and blue sequences from each $\boldsymbol{\xi}_\ell$ would be included in the pruned set $\mathtt{U}_k$, and the first input u^* of the best sequence is used to choose the state $\boldsymbol{\xi}_{\ell+1}$ for the next iteration.

1) Central Idea

Whereas the value function in (21) is defined by many sequences $\boldsymbol{\mu}_k^h \in \mathcal{U}^{h-k}$, relatively few are ever chosen by an optimal trajectory. In practice, this should enable us to replace the "large" sets of sequences $\mathcal{U}^{h-k}$ in (21) by significantly smaller subsets $\mathtt{U}_k \subseteq \mathcal{U}^{h-k}$ while achieving a near-optimal cost. We discuss here a recursive method of constructing such subsets.

Recall that the vectors $\mathbf{d}(\boldsymbol{\mu}_k^h)$ that define the cost-to-go from time k are recursively constructed via (17). This motivates us to construct the subsets $\mathtt{U}_k \subseteq \mathcal{U}^{h-k}$ such that

$$\mathtt{U}_k \subseteq \mathcal{U} \otimes \mathtt{U}_{k+1} := \{u\boldsymbol{\mu} : u \in \mathcal{U}, \boldsymbol{\mu} \in \mathtt{U}_{k+1}\},$$

where $u\boldsymbol{\mu}$ denotes a concatenation of sequences. In this way, if we have already computed $\mathbf{d}(\boldsymbol{\mu}_{k+1}^h)$ for all $\boldsymbol{\mu}_{k+1}^h \in \mathtt{U}_{k+1}$, then we are guaranteed to have what we need to compute $\mathbf{d}(\boldsymbol{\mu}_k^h)$ for all $\boldsymbol{\mu}_k^h \in \mathtt{U}_k$.

For k near the end of the horizon, the number of possible sequences $\mathcal{U}^{h-k}$ is small enough that we can take $\mathtt{U}_k$ as the full set of pre-extensions $\mathcal{U} \otimes \mathtt{U}_{k+1}$. When these sets get large, we heuristically choose a subset of $\mathcal{U} \otimes \mathtt{U}_{k+1}$ via a combination of "exploration" and "exploitation". The "exploration" phase consists of choosing a random set of sequences. In the "exploitation" phase, we use the already computed vectors $\mathbf{d}(\boldsymbol{\mu}_{k+1}^h)$ to dynamically evaluate the costs of the associated sequences $\boldsymbol{\mu}_k^h = u\boldsymbol{\mu}_{k+1}^h$ from a number of states, and we save those that perform well.

2) The Algorithm

The dynamic pruning algorithm we present is dependent on a number of parameters: a fixed capacity $L \in \mathbb{N}$ that denotes the maximum number of sequences stored in the set $\mathtt{U}_k$ at any iteration k; a set of initial states $\mathtt{X}_0 = \{x_1, \ldots, x_{M_x}\} \subseteq \mathcal{X}$ for dynamic cost evaluation; integers $M_b, M_r \in \mathbb{N}$ that denote the respective numbers of best (exploitation) and random (exploration) sequences to be saved at each iteration. These parameters should be chosen so that

$$hM_bM_x + M_r \leq L, \quad (24)$$

to ensure that $|\mathtt{U}_k| \leq L$ at every iteration as desired.

Algorithm 1 Dynamic Pruning

1: $\mathtt{U}_h \leftarrow \{\emptyset\}$ ▷ *empty sequence*
2: ▷ *Initialize parameters that define cost at final time* h
3: $\mathbf{d}_\emptyset \leftarrow \mathbf{c}_h$
4: $W_\emptyset \leftarrow Z$
5: $\beta_\emptyset \leftarrow \hat{\varepsilon}\|\mathbf{c}_h\|_q$
6: $\boldsymbol{\omega}_\emptyset \leftarrow \mathbf{z} - \hat{\varepsilon}\boldsymbol{\rho}_q(Z)$
7: **for** $k \in \{h, \ldots, 1\}$ **do**
8: **if** $n_u|\mathtt{U}_k| \leq L$ **then**
9: $\mathtt{U}_{k-1} \leftarrow \mathcal{U} \oplus \mathtt{U}_k$
10: **else**
11: Define $Q_k : \mathbb{R}^N \times \mathtt{U}_k \to \mathbb{R}$ as $Q_k(\boldsymbol{\xi}, \boldsymbol{\mu}) := \mathbf{d}_{\boldsymbol{\mu}}^T \boldsymbol{\xi}$
12: $\mathtt{U}_{k-1} \leftarrow \text{PruneStep}(\mathtt{U}_k, Q_k)$
13: ▷ *Compute parameters for cost-to-go from time* $k-1$
14: **for** $u\boldsymbol{\mu} \in \mathtt{U}_{k-1}$ **do** ▷ *where* $u \in \mathcal{U}$, $\boldsymbol{\mu} \in \mathtt{U}_k$
15: $\mathbf{d}_{u\boldsymbol{\mu}} \leftarrow A_u^T \mathbf{d}_{\boldsymbol{\mu}} + \mathbf{c}_{k-1}$
16: $W_{u\boldsymbol{\mu}} \leftarrow W_{\boldsymbol{\mu}} A_u$
17: $\beta_{u\boldsymbol{\mu}} \leftarrow \beta_{\boldsymbol{\mu}} + \hat{\varepsilon}\|\mathbf{d}_{u\boldsymbol{\mu}}\|_q + (\varepsilon_u - \hat{\varepsilon})\|\mathbf{d}_{\boldsymbol{\mu}}\|_q$
18: $\boldsymbol{\omega}_{u\boldsymbol{\mu}} \leftarrow \boldsymbol{\omega}_{\boldsymbol{\mu}} - \hat{\varepsilon}\boldsymbol{\rho}_q(W_{u\boldsymbol{\mu}}) - (\varepsilon_u - \hat{\varepsilon})\boldsymbol{\rho}_q(W_{\boldsymbol{\mu}})$
return $\mathtt{U}_0$, $\{\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}, \beta_{\boldsymbol{\mu}}, \boldsymbol{\omega}_{\boldsymbol{\mu}}\}_{\boldsymbol{\mu} \in \mathtt{U}_0}$

Algorithm 2 PruneStep($\mathtt{U}_k, Q_k$)

1: **for** $x_j \in \mathtt{X}_0$ **do**
2: $\boldsymbol{\xi}_0 \leftarrow \boldsymbol{\Psi}(x_j)$
3: **for** $\ell \in \{0, \ldots, h-1\}$ **do**
4: ▷ *Compute cost from* $\boldsymbol{\xi}_\ell$ *for each* $u\boldsymbol{\mu} \in \mathcal{U} \otimes \mathtt{U}_k$
5: **for** $u \in \mathcal{U}$ **do**
6: $\boldsymbol{\xi}_\ell^u \leftarrow A_u \boldsymbol{\xi}_\ell$
7: **for** $\boldsymbol{\mu} \in \mathtt{U}_k$ **do**
8: $J_{u\boldsymbol{\mu}} \leftarrow Q_k(\boldsymbol{\xi}_\ell^u, \boldsymbol{\mu})$
9: $\mathtt{U}_{\text{best}}^{j,\ell} \leftarrow$ Set of sequences $u\boldsymbol{\mu} \in \mathcal{U} \oplus \mathtt{U}_k$ that achieve the M_b lowest costs $J_{u\boldsymbol{\mu}}$
10: $u^* \leftarrow \arg\min_{u \in \mathcal{U}} \min_{\boldsymbol{\mu} \in \mathtt{U}_k} J_{u\boldsymbol{\mu}}$
11: $\boldsymbol{\xi}_{\ell+1} \leftarrow \boldsymbol{\xi}_\ell^{u^*}$ ▷ *next iteration starts from* $A_{u^*}\boldsymbol{\xi}_\ell$
12: $\mathtt{U}_{\text{best}} \leftarrow \bigcup_{j=1}^{M_x} \bigcup_{\ell=0}^{h-1} \mathtt{U}_{\text{best}}^{j,\ell}$
13: $\mathtt{U}_{\text{rand}} \leftarrow \{M_r \text{ random samples from } \mathcal{U} \oplus \mathtt{U}_k\}$
return $\mathtt{U}_{\text{best}} \cup \mathtt{U}_{\text{rand}}$

Algorithm 1 contains the main loop, in which sets of sequences $\mathtt{U}_k$ are iteratively constructed backwards from the end of the horizon. If needed, the PruneStep function defined by Algorithm 2 is called to construct a dynamically pruned set $\mathtt{U}_k$ that is smaller than the capacity L. The corresponding $\{\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}, \beta_{\boldsymbol{\mu}}, \boldsymbol{\omega}_{\boldsymbol{\mu}}\}_{\boldsymbol{\mu} \in \mathtt{U}_k}$ values that define the cost-to-go function are recursively updated in each iteration.

The pruning step in Algorithm 2 takes a set of input sequences $\mathtt{U}_k$ and a function Q_k that evaluates the cost of using a given input sequence from a given state, and returns a subset of $\mathcal{U} \otimes \mathtt{U}_k$. This subset is constructed via a combination of "exploitation" and "exploration". The exploitation phase, illustrated in Figure 1, is performed as follows. For each x_j in some chosen fixed set $\mathtt{X}_0 = \{x_1, \ldots, x_{M_x}\} \subseteq \mathcal{X}$, we compute the projected costs from $\boldsymbol{\xi}_0 = \boldsymbol{\Psi}(x_j)$ for each sequence in $\mathcal{U} \otimes \mathtt{U}_k$, and use the first input of the best sequence to recursively follow a near-optimal trajectory in the lifted state space. At each state along this trajectory, we save the M_b best sequences from $\mathcal{U} \otimes \mathtt{U}_k$ to include them in the pruned subset. The exploration phase chooses some number M_r of random sequences to be included in the pruned subset.

When Algorithm 1 terminates, we will have constructed a subset $\mathtt{U}_0 \subseteq \mathcal{U}^h$ and the associated $\{\mathbf{d}_{\boldsymbol{\mu}}, W_{\boldsymbol{\mu}}, \beta_{\boldsymbol{\mu}}, \boldsymbol{\omega}_{\boldsymbol{\mu}}\}_{\boldsymbol{\mu} \in \mathtt{U}_0}$. These define the following optimization problem

$$\begin{aligned} \min_{\boldsymbol{\mu} \in \mathtt{U}_0} \quad & \mathbf{d}_{\boldsymbol{\mu}}^T \hat{\boldsymbol{\Psi}}(x_0) + \beta_{\boldsymbol{\mu}} \\ \text{s.t.} \quad & W_{\boldsymbol{\mu}} \hat{\boldsymbol{\Psi}}(x_0) \leq \boldsymbol{\omega}_{\boldsymbol{\mu}}, \end{aligned}$$

which provides a tractable approximation of $J_0^\varepsilon(x_0)$ as defined by (21).

Remark 1: We have written Algorithm 2 to take the function Q_k as an input argument to emphasize that other heuristics could be used with the same algorithmic structure. For example, one could alternatively define Q_k as

$$Q_k(\boldsymbol{\xi}, \boldsymbol{\mu}) := \begin{cases} \mathbf{d}_{\boldsymbol{\mu}}^T \boldsymbol{\xi} + \beta_{\boldsymbol{\mu}}, & \text{if } W_{\boldsymbol{\mu}} \boldsymbol{\xi} \leq \boldsymbol{\omega}_{\boldsymbol{\mu}}, \\ \infty & \text{otherwise,} \end{cases}$$

which considers constraints and robustness in addition to the nominal cost. However, this has a higher computational cost compared the Q_k defined in step 11 of Algorithm 1, as it requires computing the product $W_{\boldsymbol{\mu}} \boldsymbol{\xi}_\ell^u$ at every iteration of the innermost loop in step 8 of Algorithm 2.

3) Computational Complexity

We now discuss the computational complexity of the algorithm, working from the innermost loop out. In step 8 of Algorithm 2, we require N operations to compute $Q_k(\boldsymbol{\xi}, \boldsymbol{\mu}) = \mathbf{d}_{\boldsymbol{\mu}}^T \boldsymbol{\xi}$, and we repeat this for at most L sequences $\boldsymbol{\mu} \in \mathtt{U}_k$. The computation of the product $A_u \boldsymbol{\xi}_\ell$ in step 6 requires N^2 operations. These are all repeated for n_u inputs u, for h values of ℓ, and for M_x initial states x_j. Thus, the time complexity of Algorithm 2 is $O(hn_u M_x(N^2 + LN))$.

We now turn our attention to Algorithm 1. Steps 15 and 16 require N^2 and $n_c N^2$ operations respectively to compute the products $A_u^T \mathbf{d}_{\boldsymbol{\mu}}$ and $W_{\boldsymbol{\mu}} A_u$. Steps 17 and 18 depend on the specific q-norm, but will have a linear dependence on N and so be dominated by the quadratic terms. The computations are performed for at most L sequences $u\boldsymbol{\mu} \in \mathtt{U}_{k-1}$. This loop and the call to Algorithm 12 are performed for h values of k. Thus, we conclude that the overall time complexity of Algorithm 1 is

$$O(h^2 n_u M_x(N^2 + LN) + h(1 + n_c)LN^2). \tag{25}$$

We see that the exponential dependency on the horizon h is reduced to a quadratic one. It is also quadratic in the dictionary size N, and linear in the controller capacity

L. Furthermore, the large LN^2 term has only a linear dependence on the horizon h.

A key property that enables us to achieve the complexity in (25) is that we are able to evaluate the cost of a sequence $u\boldsymbol{\mu} \in \mathcal{U} \otimes \mathtt{U}_k$ without computing the associated $\mathbf{d}_{u\boldsymbol{\mu}}$, since $\mathbf{d}_{u\boldsymbol{\mu}}^T \boldsymbol{\xi}_\ell = \mathbf{d}_{\boldsymbol{\mu}}^T A_u \boldsymbol{\xi}_\ell + \mathbf{c}_{k-1}$. This requires computing the product $A_u\boldsymbol{\xi}_\ell$ in step 6 for all $u \in \mathcal{U}$, rather than only for the optimal u^* to obtain $\boldsymbol{\xi}_{\ell+1}$ in step 11, which adds $(n_u-1)h^2M_xN^2$ operations. However, we then only perform the costly computations in steps 15 and 16 for sequences we keep in $\mathtt{U}_{k-1}$, rather than for all of $\mathcal{U} \otimes \mathtt{U}_k$, which saves $(n_u-1)h(1+n_c)LN^2$ operations. Thus, we benefit so long as $(1+n_c)L > hM_x$, which is always the case due to (24).

IV. ENTITY-BASED SYSTEMS

We now define the concept of an *entity-based system*. These are systems of the form (1) for which the state x is composed of a finite number of entities, each with their own individual state. These entities may be grouped into classes of like-entities, within which entities are assumed to behave anonymously, i.e. the state remains effectively unchanged if the entities' states are permuted. As a result, each class of like-entities can be identified with a set-valued[1] 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.

A key property of an entity-based system is that, given just the sets $\mathcal{E}_i(x)$ for each entity class $i \in \{1, \dots, n_e\}$, one can uniquely determine the system dynamics and output. In effect, the system (1) on the state space $\mathcal{X}$ can be alternatively represented by a system on an entity-based state space

$$\mathscr{X} := 2^{\mathbb{R}^{m_1}} \times \cdots \times 2^{\mathbb{R}^{m_{n_e}}},$$

where $2^{\mathbb{R}^{m_i}}$ denotes the power set of $\mathbb{R}^{m_i}$. The entity-based state in $\mathscr{X}$ is then related to the underlying state $x \in \mathcal{X}$ by the mapping $\mathscr{E} : \mathcal{X} \to \mathscr{X}$ defined as

$$\mathscr{E}(x) := (\mathcal{E}_1(x), \dots, \mathcal{E}_{n_e}(x)). \tag{26}$$

The desired property is expressed mathematically in the following definition.

Definition 1:
The system (1) is an *entity-based system* with entity classes $\mathcal{E}_1, \dots, \mathcal{E}_{n_e}$ if there exist functions $\mathscr{F} : \mathscr{X} \times \mathcal{U} \to \mathscr{X}$ and $\mathscr{G} : \mathscr{X} \to \mathbb{R}^{n_y}$ such that

$$\mathscr{E}(f(x,u)) = \mathscr{F}(\mathscr{E}(x), u), \tag{27}$$
$$y = g(x) = \mathscr{G}(\mathscr{E}(x)), \tag{28}$$

for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$, where $\mathscr{E}$ is defined as in (26).

Note that the original state x may be irrecoverable from the entity sets, i.e. the mapping $\mathscr{E}$ may be non-injective, so long as (27) and (28) hold. The commutative diagram in

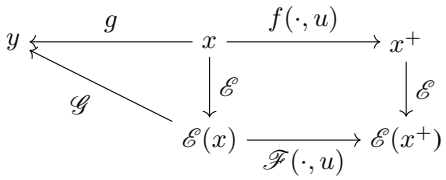

FIGURE 2. A commutative diagram illustrating the relations between system dynamics and output maps for the underlying state x and the entity-based state $\mathscr{E}(x)$.

Figure 2 illustrates the necessary relations between the state $x \in \mathcal{X}$, the entity-based state $\mathscr{E}(x) \in \mathscr{X}$, and the output $y = g(x)$.

A. DENSITY OBSERVABLES

Since the entity-based state $\mathscr{E}(x)$ is sufficient for determining the dynamics and the output at each time step, then it is sufficient to choose a dictionary $\boldsymbol{\Psi}$ that depends only on these sets. To accomplish this, we use observables that describe a spatial "density" of entities within their state space.

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

$$\phi_{ij}(x) = \sum_{\mathbf{e} \in \mathcal{E}_i(x)} w_{ij}(\mathbf{e}) \kappa_i(\mathbf{e}, \mathbf{b}_{ij}) \tag{29}$$

where w_{ij} is a chosen weighting function and κ_i is a chosen kernel function. One appropriate choice of kernel is the Gaussian radial basis kernel

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

where $\ell_i > 0$ is a chosen lengthscale parameter. We then define a "sub-dictionary" containing these observables as $\boldsymbol{\Phi}_{\mathcal{E}_i} := [\phi_{i1} \cdots \phi_{iN_i}]^T$. As illustrated in Figure 3, this resulting sub-dictionary captures in some sense a spatial "density" of the entities. The full dictionary $\boldsymbol{\Psi}$ for the Koopman model (5) can be constructed as the concatenation of the sub-dictionaries for each entity class, i.e. $\boldsymbol{\Psi} := [\boldsymbol{\Phi}_{\mathcal{E}_1}^T \cdots \boldsymbol{\Phi}_{\mathcal{E}_{n_e}}^T]^T$, and thus has size $N = \sum_{i=1}^{n_e} N_i$.

We note that, although we defined the observables ϕ_{ij} in (29) as functions of the state x, the dependency on x only appears through the entity set $\mathcal{E}_i(x)$. That is, we can write $\phi_{ij} = \widetilde{\phi}_{ij} \circ \mathcal{E}_i$ for appropriately defined $\widetilde{\phi}_{ij} : 2^{\mathbb{R}^{m_i}} \to \mathbb{R}$. The full dictionary $\boldsymbol{\Psi}$ can similarly be written as $\boldsymbol{\Psi} = \widetilde{\boldsymbol{\Psi}} \circ \mathscr{E}$ for appropriately defined $\widetilde{\boldsymbol{\Psi}} : \mathscr{X} \to \mathbb{R}^N$.

While this construction can be applied to any collection of basis centers $\{\mathbf{b}_{ij}\}_{j=1}^{N_i}$, choosing them strategically can allow for better coverage of the most relevant regions of the state space. This can be done in a data-driven manner based on a set of states $\{x_\ell\}_{\ell=1}^{M_K}$ collected across one or more trajectories, akin to the set of data snapshots (8) used in EDMD. We choose the centers for each entity class $i \in \{1, \dots, n_e\}$ via K-means clustering on the entity states

[1] In some systems, it may be possible for two or more entities to exist in identical states to each other, in which case the mappings $\mathcal{E}_i$ should instead be considered to be multiset-valued so that multiplicities are recorded.

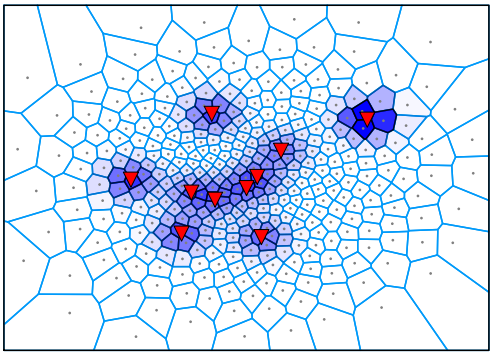

FIGURE 3. An example illustration of how the observables defined in (29) describe a "density" of the entities. Entities are marked by triangles, and are overlaid on a Voronoi tessellation of basis centers $\{\mathbf{b}_{ij}\}_{j=1}^{N_i}$. The cell of each center $\mathbf{b}_{ij}$ is shaded proportionally to the value of the corresponding observable $\phi_{ij}(x)$.

reached by this set, i.e.

$$\{\mathbf{b}_{ij}\}_{j=1}^{N_i} = \underset{K=N_i}{K\text{-means}} \left(\bigcup_{\ell=1}^{M_k} \mathcal{E}_i(x_\ell) \right), \quad \forall i \in \{1, \dots, n_e\}.$$

In this way, we avoid choosing centers in regions where entities are unlikely to go in order to capture finer details in regions where entities spend most of their time.

B. PRODUCTS OF DENSITIES

The density observables we have defined may be insufficient for accurately describing the dynamics of some certain systems via a linear Koopman approximation. In particular, interactions between different entities can be difficult to capture. Such issues can be addressed by introducing observables that depend on multiple classes of entities. Our approach is to construct such observables via products of the separate entity densities.

For a pair of entity classes $\mathcal{E}_i$ and $\mathcal{E}_k$, we define the product density $\mathbf{\Phi}_{\mathcal{E}_i \otimes \mathcal{E}_k} : \mathcal{X} \to \mathbb{R}^{N_i N_k}$ between them as

$$\mathbf{\Phi}_{\mathcal{E}_i \otimes \mathcal{E}_k}(x) := \mathbf{\Phi}_{\mathcal{E}_i}(x) \otimes \mathbf{\Phi}_{\mathcal{E}_k}(x)$$

where $\otimes$ denotes the Kronecker product. The observable $[\mathbf{\Phi}_{\mathcal{E}_i \otimes \mathcal{E}_k}(x)]_{N_i(j-1)+\ell} = \phi_{ij}(x)\phi_{k\ell}(x)$ of this product density then indicates whether there are entities from both $\mathcal{E}_i$ and $\mathcal{E}_k$ that are near to the basis centers $\mathbf{b}_{ij}$ and $\mathbf{b}_{k\ell}$ respectively. We remark that, since Kronecker products of even moderately sized vectors can be large, it may be infeasible to include the full product density in the Koopman dictionary. In such cases, we may choose to use only a subset of the full Kronecker product.

Example 1:

We provide here an example demonstrating how product densities are able to capture dynamics that the separate densities cannot capture on their own. Consider a system consisting of two entity classes $\mathcal{E}_1, \mathcal{E}_2 : \mathcal{X} \rightrightarrows \mathbb{N}$, with dynamics

$$\mathcal{E}_1(x^+) = \{e + 1 : e \in \mathcal{E}_1(x)\}, \tag{30a}$$
$$\mathcal{E}_2(x^+) = \mathcal{E}_2(x) \setminus \mathcal{E}_1(x), \tag{30b}$$

The behavior of this system is illustrated in Figure 4, where $\mathcal{E}_1(x)$ and $\mathcal{E}_2(x)$ contain the current positions of entities in each class. In this system, $\mathcal{E}_1$ entities move with constant velocity while $\mathcal{E}_2$ entities remain stationary. If entities from the two classes meet, they interact such that the $\mathcal{E}_2$ entity is "destroyed". This type of interaction appears, for example, in arcade games that involve firing projectiles toward enemies.

Suppose we construct densities as in (29) using a Kronecker delta kernel (defined as $\kappa_\delta(x, y) = 1$ if $x = y$, and 0 otherwise) and integer basis centers $\mathbf{b}_{ij} = j$, so that

$$\begin{aligned}\phi_{ij}(x) &:= \sum_{e \in \mathcal{E}_i(x)} \kappa_\delta(e, j) \\ &= \begin{cases} 1, & j \in \mathcal{E}_i(x), \\ 0, & \text{otherwise}, \end{cases} \qquad \forall i \in \{1, 2\},\ j \in \mathbb{N}.\end{aligned}$$

As a minimal example we consider the dictionary $\mathbf{\Psi} = [\phi_{11} \ \ \phi_{21}]^T$ and its behavior on three specific states as described by Table 1. In particular, we note that $\mathbf{\Psi}(x_3) = \mathbf{\Psi}(x_1) + \mathbf{\Psi}(x_2)$, and so for any matrix A such that $\mathbf{\Psi}(x^+) = A\mathbf{\Psi}(x)$ for all x, we would have

$$\begin{aligned}\mathbf{\Psi}(x_3^+) &= A\mathbf{\Psi}(x_3) \\ &= A\mathbf{\Psi}(x_1) + A\mathbf{\Psi}(x_2) = \mathbf{\Psi}(x_1^+) + \mathbf{\Psi}(x_2^+).\end{aligned}$$

However, this is a contradiction since we have that $\mathbf{\Psi}(x_3^+) \neq \mathbf{\Psi}(x_1^+) + \mathbf{\Psi}(x_2^+)$, and so we conclude that no such A exists.

Now, we augment $\mathbf{\Psi}$ with a product density to obtain a new dictionary $\mathbf{\Psi}_{\text{aug}} := [\phi_{11} \ \ \phi_{21} \ \ \phi_{11}\phi_{21}]^T$. For this dictionary, we have

$$\mathbf{\Psi}_{\text{aug}}(x_3) = \begin{bmatrix}1\\1\\1\end{bmatrix} \neq \begin{bmatrix}1\\0\\0\end{bmatrix} + \begin{bmatrix}0\\1\\0\end{bmatrix} = \mathbf{\Psi}_{\text{aug}}(x_1) + \mathbf{\Psi}_{\text{aug}}(x_2),$$

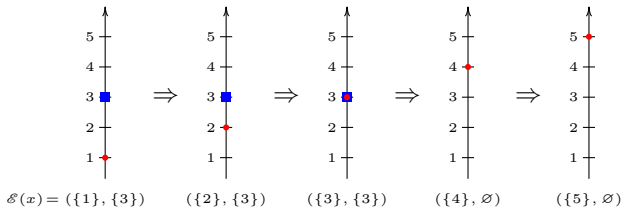

FIGURE 4. An illustration of a possible state trajectory for the example system described by (30). The red circle denotes an $\mathcal{E}_1$ entity, while the blue square denotes an $\mathcal{E}_2$ entity.

TABLE 1. Three states for the system in Example 1, showing the entity-based state $\mathscr{E}(x)$ and lifted state $\mathbf{\Psi}(x)$ at the current and next time steps. No matrix A satisfies $\mathbf{\Psi}(x^+) = A\mathbf{\Psi}(x)$ for all x.

	$\mathscr{E}(x)$	$\mathbf{\Psi}(x)^T$	$\mathscr{E}(x^+)$	$\mathbf{\Psi}(x^+)^T$
x_1	$(\{1\}, \varnothing)$	[1 0]	$(\{2\}, \varnothing)$	[0 0]
x_2	$(\varnothing, \{1\})$	[0 1]	$(\varnothing, \{1\})$	[0 1]
x_3	$(\{1\}, \{1\})$	[1 1]	$(\{2\}, \varnothing)$	[0 0]

and so do not encounter the same contradiction as before. Indeed, we can verify that the matrix

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

satisfies $\mathbf{\Psi}_{\text{aug}}(x^+) = A\mathbf{\Psi}_{\text{aug}}(x)$ for all x.

V. NUMERICAL RESULTS

We present here numerical results for two systems. First is a relatively simple game implemented in Julia that we refer to as *MultiPong*, which we can fully customize to test different aspects of our methods. Second is the Atari 2600 game *Assault*, which presents a more complex challenge.

A. MULTIPONG

MultiPong is an evolution of the simple Pong game considered in [17], which is itself inspired by the Atari 2600 game *Pong*. The game consists of a 1×1 square box containing one or more "balls", denoted by points with associated velocity vectors, and one or more "paddles" denoted by vertical line segments with length 0.1. The "Multi" part of the game's name refers to the presence of multiple balls and paddles, which contrasts it from the games that inspire it.

The balls move within the box, colliding elastically with the walls and the paddles. The paddles are located at a single fixed x-coordinate, and may move between twelve fixed y positions that range evenly from 0.05 to 0.95 (denoting the midpoint of the paddle), as shown in Figure 5. In instances with only one paddle, the player chooses at each time step to either move the paddle up, down, or leave it in place. Attempting to move up from the top position or down from the bottom position has no effect. The case with multiple paddles will be discussed later.

The objective of the game is to keep the balls to the left of the paddle for as much time as possible. If a ball does cross to the right of the paddle, the game continues with the ball continuing to move within the region to the right of the paddle. Balls on the right side can also be deflected backward, and so the player should try to allow these balls to cross back into the left region. We use a time-invariant cost function $c(x)$ that counts the number of balls that are to the right of the paddle. One episode of the game ends after 100 time steps have elapsed.

To evaluate the efficacy of our algorithm in isolation from errors in the Koopman approximation, we first generate initial states in specific configurations such that the balls will remain confined to a finite set of states, for which we can obtain an exact Koopman representation. Letting $n_x, n_y \in \mathbb{N}$, we fix the initial x positions and velocities for the balls as

$$x = \frac{1}{2n_x}, \qquad v_x = \frac{1}{n_x}, \tag{31}$$

and we randomly choose initial y positions and velocities

$$y \in \left\{ \frac{2k-1}{2n_y} \right\}_{k=1}^{n_y}, \qquad v_y \in \left\{ -\frac{1}{n_y}, \frac{1}{n_y} \right\}.$$

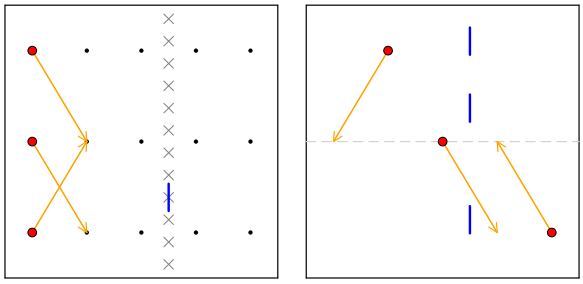

FIGURE 5. **Left: One possible initial state with $n_x = 5$, $n_y = 3$, and three balls. The large dots denote the balls' position, and the arrows their velocities. The vertical line segment denotes the paddle, and the crosses denote the positions it moves between. From this state, future positions of the balls will be confined to a finite grid which is marked by small dots; Right: One possible state with three balls and three paddles. The dashed line denotes the separation between the top and bottom control regions, so that the paddles above the line move in the same direction, while the bottom paddle can move in a different independent direction.**

The positions reachable by the ball then form an $n_x \times n_y$ grid as illustrated in the left of Figure 5, as long as the x-coordinate of the paddle is chosen to be exactly halfway between two consecutive x-coordinates of this grid. For each n_x, we choose the nearest such coordinate to $x = 2/3$.

The total number of possible states for each ball is then $4n_xn_y$, since there are n_xn_y reachable positions, each with one of four possible diagonal velocities. Since there are twelve possible positions for each paddle, then the total number of possible states for a game with one ball and one paddle is $48n_xn_y$.

We define two entity classes for the MultiPong system. The first class $\mathcal{E}_1$ consists of the balls, each with a *(position, velocity)* state in $\mathbb{R}^4$, and the second class $\mathcal{E}_2$ consists of the paddles, each with a state in $\mathbb{R}$. The basis centers for each entity class are chosen respectively as the set of the $4n_xn_y$ ball states in $\mathbb{R}^4$, and the set of the 12 paddle states in $\mathbb{R}$.

We fix $n_y = 3$ throughout for all examples. The vertical velocity of the ball is thus much greater than that of the paddle, and so future predictions of the balls' trajectory are necessary to know where the paddle should be moved toward. Except where otherwise noted, all Koopman-based controllers are constructed via Algorithm 1 with horizon $h = 20$ and with controller capacity $L = 10{,}000$, $M_r = 5{,}000$ randomly chosen sequences per time step, $M_b = 25$ sequences chosen per visited lifted state, and the initial state set $\mathtt{X}_0$ is taken as a set of 10 random initial states.

1) Control with Exact Koopman Models

For games with the parameter n_x ranging from 5 to 15, we compare two Koopman dictionaries, with and without product densities, defined as

$$\mathbf{\Psi}_{\text{separate}} := [\mathbf{\Phi}_{\mathcal{E}_1}^T \ \mathbf{\Phi}_{\mathcal{E}_1}^T]^T \quad \text{and} \quad \mathbf{\Psi}_{\text{prod}} := \mathbf{\Phi}_{\mathcal{E}_1 \otimes \mathcal{E}_2},$$

respectively. For each dictionary, we generate the set of data snapshots (8) for EDMD by applying each of the three inputs $u \in \mathcal{U}$ to each of the $48n_xn_y$ states for a game with one ball and one paddle. Since this total number of states is finite,

then it is guaranteed that an exact Koopman model exists for some finite dictionary, and indeed we find that $\mathbf{\Psi}_{\text{prod}}$ achieves this (to within numerical precision).

Although the Koopman dictionaries and EDMD models are learned only for a game with one ball, the entity-based framework allows us to re-use them for games with any number of balls. For the dictionary $\mathbf{\Psi}_{\text{prod}}$, the learned Koopman model still predicts the dynamics exactly (within numerical precision) when additional balls are added.

For reference, we compare with two non-Koopman policies. The first is a "greedy" policy that moves the paddle toward the ball's current y-coordinate. When multiple balls are present, it prioritizes whichever is closest in Euclidean distance. When all balls are to the right of the paddle, it instead moves away from the closest ball's y-coordinate. The second is a "predictive" policy that uses exact knowledge of the nonlinear dynamics to compute the exact y-coordinate where the ball will cross the paddle's x-coordinate, and moves toward that point. When multiple balls are present, it prioritizes whichever will pass the paddle line first. When all balls are to the right of the paddle, it instead moves away from the point where the closest ball will cross the paddle line right-to-left. We remark that this policy is nearly optimal in the single-ball case, but is suboptimal for multiple balls as it lacks sophisticated prioritization.

We compare the performance of Koopman-based policies constructed via Algorithm 1 to that of the greedy and predictive policies described above. Recall that the pruning algorithm includes random exploration (line 13 in Algorithm 2), and as such we consider the average performance of 10 constructed policies for each EDMD model. Each policy is applied in feedback from each possible initial state. The results of these are shown in Figure 6. Note that due to the discrete time steps and our discretization of the states, even a small change to n_x can result in very different closed-loop trajectories. The Koopman policy without products performs poorly, achieving results on par with the greedy policy. The Koopman policy with products performs well, matching the predictive policy for games with one ball, and achieving better performance for games with three balls.

We also evaluate the effect of the controller capacity L in Algorithm 1. For $n_x \in \{5, 10, 15\}$, we test ten Koopman policies each for varying L (and varying M_b and M_r in proportion to L so that (24) is satisfied), applied in feedback from each possible initial state. The results are shown in Figure 7, from which we see that good performance can be achieved even with relatively small controllers. The 3 ball game with $n_x = 10$ is the only case where we see meaningful improvement as the controller capacity increases, which plateaus beyond $L = 10{,}000$ sequences.

2) Introducing Model Errors

We now investigate how well our methods perform when we are unable to obtain an exact Koopman model. We do this by introducing small perturbations to the initial states, which will cause the ball to gradually diverge from the nominal grid points. Moreover, we can control the size of these errors by changing the scale of these perturbations. Larger perturbations will result in less accurate Koopman approximations.

For a chosen $\delta > 0$, we generate initial states as before, but we now multiply the initial x and y velocities respectively by two values chosen uniformly from the interval $[1-\delta, 1+\delta]$. Figure 8 illustrates sample trajectories for three different values of δ. Note that even small δ can have a large effect as the deviation from the unperturbed trajectory accumulates over time.

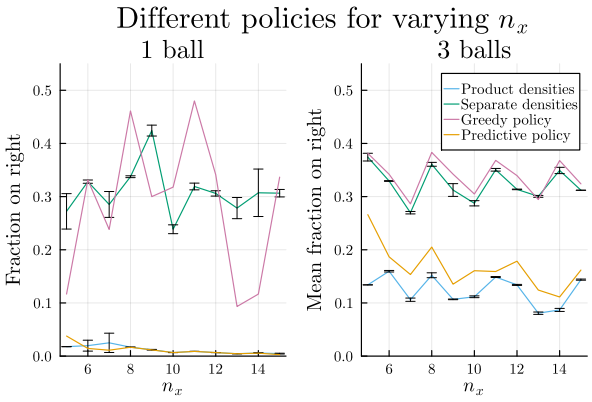

FIGURE 6. Performance of different control policies when varying the n_x chosen as the initial x position and velocity in (31). The size of dictionaries for the learned Koopman model are directly proportional to n_x.

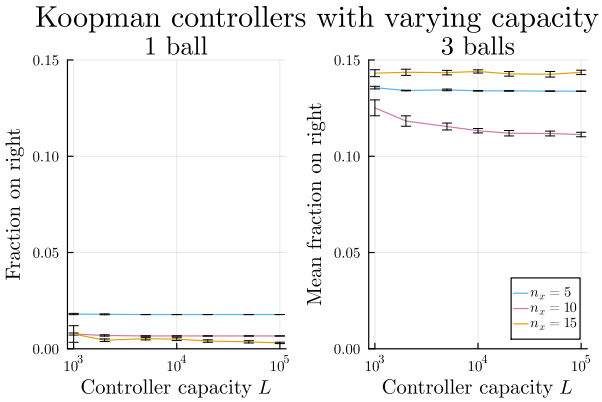

FIGURE 7. Performance of different Koopman policies with different controller capacities used for Algorithm 1.

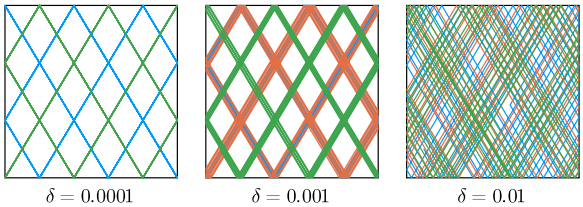

FIGURE 8. Sample ball trajectories for three different values of the perturbation scale δ over a horizon of 100 time steps.

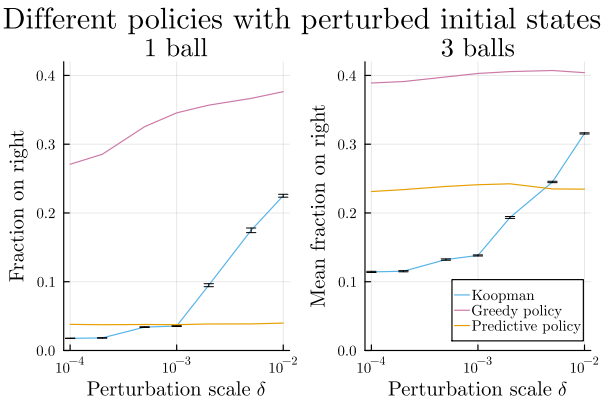

FIGURE 9. **Performance of different control policies with initial states perturbed by an amount proportional to δ. This introduces error in the Koopman model that grows with δ. The sample trajectories in Figure 8 span the range of tested δ.**

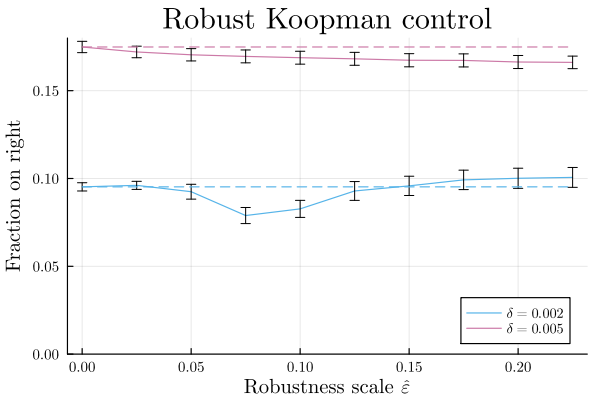

FIGURE 10. **Performance of robust Koopman policies as described in Theorem 2. The bound $\hat{\varepsilon}$ on the 1-norm error of the initial lifted state is varied. The dashed line shows the result without robustness (i.e. $\hat{\varepsilon} = 0$) for ease of comparison. Error bars denote 95% confidence intervals.**

We vary the perturbation scale δ while fixing $n_x = 5$, and learn a new Koopman model for each δ. We construct each dictionary via K-means clustering over 1,000 episodes, and perform EDMD over another 1,000 episodes (yielding 100,000 total snapshots). The performance of each control policy is shown in Figure 9. Each data point for the Koopman-based policies represents the mean performance of 100 constructed controllers, each evaluated across 200 initial states. The greedy and predictive policies are each evaluated on 20,000 random initial states. For sufficiently small δ, the Koopman controllers maintain good performance. For larger perturbations, the ball no longer remains close to the nominal grid points, resulting in increased prediction errors and deteriorated performance.

3) Control with Robustness

We now show how our robust control policy described in Theorem 2 can be used to improve performance in the presence of modeling errors. For $\delta \in \{0.002, 0.005\}$, we test varying bounds $\hat{\varepsilon}$ on the 1-norm error of the initial lifted state. We use the same dictionaries and Koopman models learned in the previous subsection.

The results are shown in Figure 10. Each data point represents the mean performance of 100 constructed controllers, each evaluated across 200 initial states. We see that robustness leads to increased performance in both cases. For $\delta = 0.002$, we see a significant improvement for a relatively narrow range of $\hat{\varepsilon}$. For $\delta = 0.005$, we see a smaller but still significant improvement that is present for a wide range of $\hat{\varepsilon}$.

4) Control with Constraints

We now discuss MultiPong instances with multiple paddles, as illustrated in the right of Figure 5. We take our inputs now as pairs $(u_{\text{top}}, u_{\text{bottom}}) \in \{\text{up}, \text{down}, \text{stay}\}^2$, where u_{top} and u_{bottom} are applied to paddles in the top and bottom zones respectively. In this way, the paddles move synchronously if they are in the same zone, but independently if they are in different zones. With this set of inputs, it can be seen that the system dynamics satisfy the requirement (27) for an entity-based system.

We also demonstrate here the usefulness of constraints. Note that it is redundant for two paddles to occupy the same position. Furthermore, if two paddles ever do occupy the same position, the zone-based control means they can never separate from each other for the remainder of the episode. As such, we implement a constraint that there should be no more than one paddle present in any one location. With twelve possible positions for the paddle, this leads to a set of $n_c = 12$ constraints.

Figure 11 shows the performance both with and without constraints in a system with varying numbers of balls and paddles. Each data point represents the mean performance of ten controllers, each tested across 200 random initial states. Compared to previous tests, we use here a reduced horizon of $h = 12$. However, due to the increase in the number of inputs from $n_u = 3$ to $n_u = 9$, the total number of possible input sequences rises to 9^{12} (about 282 billion). We use the same controller capacity $L = 10{,}000$ for pruning, so that this results in a smaller fraction of possible sequences being considered.

We compare unconstrained controller with constrained controllers constructed via Algorithm 1. For the constrained case, we additionally test pruning with the constrained Q_k as described in Remark 1. Figure 11 shows that the inclusion of constraints leads to significantly better performances in most cases. No significant difference is seen when pruning with constrained value functions Q_k. Note that the improvement in the case with three balls and two paddles is smaller than for other cases, since the two paddles are insufficient for reliably deflecting all three balls even with perfect play.

B. ASSAULT

We now consider a more complex example, namely the Atari 2600 game *Assault*, which we interface with the game via the Arcade Learning Environment introduced by [21]. Due

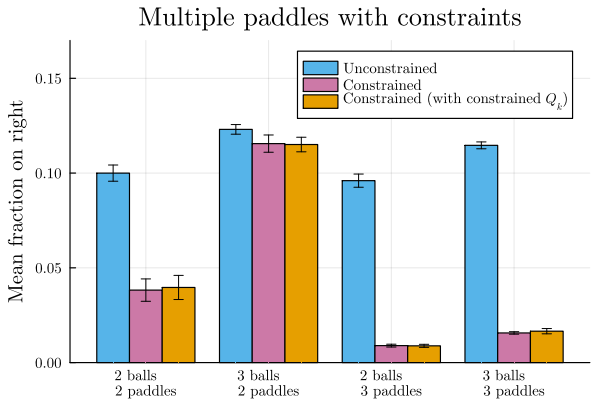

FIGURE 11. **Performance of Koopman controllers in a system with multiple balls and multiple paddles. We implement constraints that enforce that we never have more than one paddle in the same position.**

to presenting a variety of challenging and independently developed tasks, such arcade games have been commonly used as a benchmark in reinforcement learning [22].

In *Assault*, the player controls a ship near the bottom of the screen, and wishes to defeat a number of enemies that appear in the middle of the screen. The player has six inputs: move left, move right, shoot left, shoot right, shoot up, or do nothing. When a projectile shot by the player hits an enemy, that enemy is killed and the player scores a certain number of points. Conversely, when a projectile shot by an enemy hits the player, the player is killed. A gauge at the bottom right increases when the player shoots, but gradually decreases otherwise. If the player shoots when the gauge is full, the player is killed. The objective of the game is to score as many points as possible before being killed four times.

We identify four relevant classes of entities as listed in Table 2 and illustrated in Figure 12. We determine which class each pixel should belong to based on color and screen position, and compute the density observables defined in (29) directly from the pixel coordinates. In this way, we obtain a representation of the state without needing to perform image segmentation. We construct product densities between the player ship and enemies, and between the player projectile and enemies. Since we can directly measure only the position of the entities, we also include time-delays of the separate densities $\mathbf{\Phi}_{\mathcal{E}_i}$ to capture information on their velocities. The full Koopman dictionary used is

$$\mathbf{\Psi}(x_k, x_{k-1}) := \begin{bmatrix} \mathbf{\Phi}_{\text{separate}}(x_k) \\ \mathbf{\Phi}_{\text{separate}}(x_{k-1}) \\ \mathbf{\Phi}_{\text{products}}(x_k) \end{bmatrix} \tag{32}$$

where

$$\mathbf{\Phi}_{\text{separate}}(x) := \begin{bmatrix} \mathbf{\Phi}_{\mathcal{E}_1}(x) \\ \mathbf{\Phi}_{\mathcal{E}_2}(x) \\ \mathbf{\Phi}_{\mathcal{E}_3}(x) \\ \mathbf{\Phi}_{\mathcal{E}_4}(x) \\ \mathbf{\Phi}_{\mathcal{E}_5}(x) \end{bmatrix}, \quad \mathbf{\Phi}_{\text{products}}(x) = \begin{bmatrix} \mathbf{\Phi}_{\mathcal{E}_1 \otimes \mathcal{E}_3}(x) \\ \mathbf{\Phi}_{\mathcal{E}_1 \otimes \mathcal{E}_4}(x) \\ \mathbf{\Phi}_{\mathcal{E}_2 \otimes \mathcal{E}_3}(x) \end{bmatrix}$$

The game is structured into a series of levels, each containing different types of enemies. Shooting left or right

TABLE 2. **Entity classes used for the game *Assault*. Also shown is the number of basis centers N_i used for each entity class.**

Entity class	Description	N_i
$\mathcal{E}_1$	Player ship	12
$\mathcal{E}_2$	Player projectiles	24
$\mathcal{E}_3$	Enemy ships	20
$\mathcal{E}_4$	Enemy projectiles	12
$\mathcal{E}_5$	Attack gauge	16

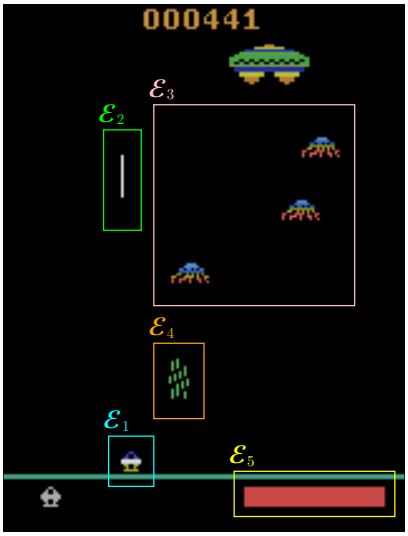

FIGURE 12. **An example screen from the game *Assault* that shows entities from the four different entity classes listed in Table 2.**

is only needed from the Level 4 onward, which is very rarely reached by the random play we use for training. As such, we exclude these inputs from training to obtain better Koopman models for the inputs that are relevant for the early levels that are explored during training. We run 1,000 episodes with random play, generating a total of 1.3 million data snapshots (8) for EDMD.

We consider the performance of control policies constructed via the dynamic pruning described in Algorithm 1, comparing a Koopman model learned for the dictionary (32) (which includes product densities) and the Koopman model from [17] (which uses only separate densities, but defined on a greater number of basis centers). Due to randomness in the "exploration" part of the pruning algorithm, different control policies may be obtained for the same Koopman model, and so we compare the mean scores of many such control policies. The results are shown in Figure 13. We find that, although the best controllers obtained for each dictionary perform similarly, the inclusion of product densities leads to consistently better policies.

VI. CONCLUSION

The Koopman operator provides an approximate linear representation to a nonlinear system, and so allows us to solve a nonlinear optimal control problem via a linear one. We showed how the optimal cost-to-go of the linear problem has a piecewise linear form on the lifted state space, which

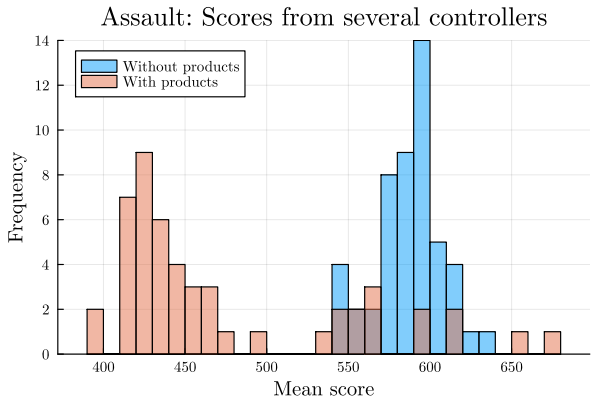

FIGURE 13. Comparison of multiple controllers constructed via the dynamic pruning described in Algorithm 1, for two different Koopman models. Due to randomness in the algorithm, different controllers may be obtained for the same Koopman model. We construct 50 such controllers for each Koopman model, and record the mean score achieved by each controller as tested over 50 episodes.

is retained in a robust version where we minimize the worst-case cost with respect to bounded errors. Due to the combinatorial nature of the problem, the number of possible input sequences grows exponentially in the problem horizon, and as such we propose a heuristic pruning algorithm that constructs a much smaller set of sequences to search over.

In addition to this generally applicable control approach, we focused on representations specific to a class of "entity-based systems" that we define. For such systems, we constructed observables that capture in some sense a spatial density of entities within their state space. We additionally introduced observables that denote a product density between two entity classes, which can accurately capture dynamics aren't captured by separate densities on their own. This is supported by the numerical results we presented on two different entity-based systems.

In our MultiPong example, we found that Koopman representations using product densities remained accurate when multiple balls were present, but that inaccuracies arose for multiple paddles. Future work could investigate when product densities are sufficient, and to determine other functions of densities that are sufficient when products are not. However, for the purpose of optimal control we remark that accuracy is not strictly necessary. Rather, it is sufficient to determine only relative orderings, i.e. whether one sequence achieves a higher cost or a lower cost than another. Thus, it may be that alternative approaches could exploit this fact to attain better control performance without needing to improve model accuracy.

REFERENCES

[1] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, pp. 315–318, May 1931.

[2] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, Dec. 2012.

[3] A. Mauroy, I. Mezić, and Y. Susuki, eds., *The Koopman Operator in Systems and Control*. 2020.

[4] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman theory for dynamical systems," *SIAM Review*, vol. 64, pp. 229–340, May 2022.

[5] Y. Lan and I. Mezić, "Linearization in the large of nonlinear systems and Koopman operator spectrum," *Physica D: Nonlinear Phenomena*, vol. 242, pp. 42–53, Jan. 2013.

[6] 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)*, pp. 7733–7739, Dec. 2019.

[7] 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. 112226, June 2025.

[8] 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, pp. 1307–1346, Dec. 2015.

[9] M. Korda and I. Mezić, "On convergence of extended dynamic mode decomposition to the Koopman operator," *Journal of Nonlinear Science*, vol. 28, pp. 687–710, Apr. 2018.

[10] 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, pp. 59–87, Feb. 2021.

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

[12] M. Korda, Y. Susuki, and I. Mezić, "Power grid transient stabilization using Koopman model predictive control," *IFAC-PapersOnLine*, vol. 51, pp. 297–302, Jan. 2018.

[13] 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, p. eadd6864, Aug. 2023.

[14] 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, pp. 4369–4376, July 2021.

[15] 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)*, pp. 7350–7356, May 2021.

[16] M. Haseli and J. Cortés, "Modeling Nonlinear Control Systems via Koopman Control Family: Universal Forms and Subspace Invariance Proximity," July 2025.

[17] 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)*, pp. 6006–6013, Dec. 2023.

[18] J. Hespanha and K. Çamsari, "Markov Chain Monte Carlo for Koopman-Based Optimal Control," *IEEE Control Systems Letters*, 2024.

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

[20] 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, pp. 529–533, Feb. 2015.

[21] 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, June 2013.

[22] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, Mar. 2018.