Learning Koopman Representations for Hybrid Systems

Craig Bakker, Arnab Bhattacharya, Samrat Chatterjee, Casey J. Perkins, and Matthew R. Oster

Abstract—The Koopman operator lifts nonlinear dynamical systems into a functional space of observables, where the dynamics are linear. In this paper, we provide three different Koopman representations for hybrid systems. The first is specific to switched systems, and the second and third preserve the original hybrid dynamics while eliminating the discrete state variables; the second approach is straightforward, and we provide conditions under which the transformation associated with the third holds. Eliminating discrete state variables provides computational benefits when using data-driven methods to learn the Koopman operator and its observables. Following this, we use deep learning to implement each representation on two test cases, discuss the challenges associated with those implementations, and propose areas of future work.

I. INTRODUCTION

For a given dynamical system \( \dot{x} = f(x) \), \( x \in \mathbb{R}^n \), with flow \( F^t(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n \), the set of Koopman operators (KOs) \( K^t \) forms a continuous semigroup acting on functions \( g(x) \):

\[
(K^t g)(x) = g(F^t(x)).
\]

The KO lifts the (nonlinear) state-space dynamics into a functional space of observables, where the dynamics are linear. A set of functions \( S \) is invariant under the KO if \( K^t g \in S \forall g \in S \). Invariance is computationally important because if \( S \) is finite-dimensional with a specified basis (of observables), the projection of \( K^t \) onto \( S \) becomes a matrix \( K^t \) that depends on the chosen basis. In practice, \( K^t \) (or \( K \), assuming a fixed time step) is typically calculated on an (approximately) invariant functional subspace using time series data. One approach specifies a dictionary of basis functions (e.g., polynomials, radial basis functions) and solves for \( K \) as a linear regression [1], [2]. Another approach uses neural networks to learn a basis from the data [3].

In computational studies, it is often valuable to include the state variables as observables so that observable space trajectories map uniquely to state space trajectories. If the observables are not state-inclusive, the mapping from observables to state variables may be difficult to calculate or non-unique. State-inclusive observables can impose additional constraints on the Koopman representation in multi-modal systems, but it is still possible to use these observables in such cases [4].

Various KO formulations for dynamical systems with control terms have been developed. Brunton et al. consider a small system with an analytical finite Koopman representation and produce a Linear Quadratic Regulator feedback control law [5]. Small changes in the underlying dynamical system, however, produce a control system that is no longer Koopman invariant. Korda and Mezić [2] combine the dictionary-of-functions approach with Model Predictive Control (MPC) and a receding time horizon approach. As shown in Bakker et al. [6], though, not all Koopman control formulations in the literature are internally consistent (i.e., they may violate the chain rule of calculus).

Almost all of the systems considered in the Koopman literature have continuous dynamics and controls. That is to say, the state and control variables are continuous, and even the discrete-time implementations are based on continuous trajectories sampled at discrete (and fixed) time intervals. Budišić et al. [7] allude briefly to the possibility of Koopman dynamics on discrete variables but do not discuss it in detail. Hanke et al. [8] discretize the control variable in a dynamical system to learn a Koopman representation for each discretized control state; control turns into switching. The underlying system, however, is fully continuous. Govindarajan et al. also consider an otherwise continuous system with impulse dynamics [9], but there are no discrete state variables in their hybrid pendulum example.

Identifying finite-dimensional sets of Koopman invariant observables is non-trivial when the observables’ domains are continuous. The increased difficulty of finding such sets with a mixture of continuous- and discrete-valued functions has likely been a barrier to the development and use of hybrid Koopman representations. However, many real-world systems of interest are not fully continuous. Applying the KO to hybrid systems could provide benefits comparable to those already demonstrated in continuous systems.

In this paper, we present three different Koopman representations for hybrid systems. The first is specific to switched systems, and the second and third preserve the original hybrid dynamics while eliminating the discrete state variables; the second approach is straightforward, and we provide conditions under which the transformation associated with the third holds. Eliminating discrete state variables provides computational benefits when using data-driven methods to learn the Koopman operator and its observables. Following this, we leverage deep learning to produce data-driven implementations of the representations on two test problems and computationally demonstrate their performance. These demonstrations show that each Koopman representation can capture the dynamics of the underlying hybrid system. Based on the computational demonstrations, we discuss the challenges associated with learning hybrid Koopman representations and suggest avenues for future research in this area.
II. Koopman Representations of Hybrid Dynamical Systems

A general discrete-time hybrid system with state variables \(x\) and \(y\) and control variables \(u\) and \(z\) is

\[
\begin{align*}
x_{k+1} &= f(x_k, y_k, u_k, z_k) \quad \text{(2)} \\
y_{k+1} &= g(x_k, y_k, u_k, z_k), \quad \text{(3)}
\end{align*}
\]

where \(x \in \mathbb{R}^n\) and \(u \in \mathbb{R}^m\) are continuous, while \(y \in \mathbb{Z}^p\) and \(z \in \mathbb{Z}^q\) are discrete. We consider the discrete-time case because most data-driven approaches rely on discretely sampled trajectory data. For the proposed Koopman representations of these systems, we will use variations on the form

\[
\psi_x(x_{k+1}) = K_x \psi_x(x_k) + K_{xu} \psi_{xu}(x_k, u_k),
\]

where \(K_x\) and \(K_{xu}\) are the finite KO approximations and \(\psi_x(x)\) and \(\psi_{xu}(x, u)\) are KO observables. For more on the use of this form, see Bakker et al. [6]. Uncontrolled systems then become a special case where \(K_{xu} = 0\). In principle, we could extend this to

\[
\psi_x(x_{k+1}, y_{k+1}) = K_x \psi_x(x_k, y_k) + K_{xu} \psi_{xu}(x_k, y_k, u_k, z_k)
\]

for a hybrid system and treat \(y\) no differently than \(x\). However, such a system may have difficulty in accounting for the discrete nature of \(y\) – especially if \(\psi_x\) is state-inclusive. On the one hand, identifying an (approximately) invariant functional subspace with continuous and discrete domains may be very difficult. On the other hand, using a continuous relaxation of \(y\) to identify its correct discrete value would actually change the dynamics in [2–5]. Additional structure is necessary to ensure that the discrete-valued \(y_{k+1}\) can always be recovered uniquely from \(\psi_x(x_{k+1}, y_{k+1})\). We therefore propose three different KO representations.

A. Switched Systems

Switched systems are a special case of hybrid systems with the switched states’ dynamics indexed by \(\lambda \in \mathbb{N}\):

\[
\begin{align*}
x_{k+1} &= f_{\lambda_k}(x_k, u_k) \quad \text{(6)} \\
\lambda_{k+1} &= g(x_k, u_k, \lambda_k). \quad \text{(7)}
\end{align*}
\]

Their structure allows for a simpler KO representation than hybrid systems in general:

\[
\begin{align*}
\psi_x(x_{k+1}) &= K_x^{\lambda_k} \psi_x(x_k) + K_{xu}^{\lambda_k} \psi_{xu}(x_k, u_k) \quad \text{(8)} \\
\lambda_{k+1} &= g(x_k, u_k, \lambda_k). \quad \text{(9)}
\end{align*}
\]

\(\lambda\) now indexes a set of \(K_x\) and \(K_{xu}\) matrices (i.e., distinct \(K_x^{\lambda_k}\) and \(K_{xu}^{\lambda_k}\) matrices for each value of \(\lambda\)). Different switched states use the same set of observables instead of having different sets of observables for each state, and there is good reason for this. The learned KO observables represent a basis for an (approximately) invariant KO subspace \(S\); the finite representation of the KO on \(S\) (i.e., \(K_x\)) depends on \(S\) and maps \(S\) to itself. A different set of observables would span a different invariant subspace \(S'\) and therefore have a different KO representation. Trying to use the KO to map from one set to the other could therefore not use either finite representation consistently. There may be ways to circumvent this problem, but for the purpose of this paper, and for the sake of simplicity, we will not attempt to do so here.

B. General Hybrid Systems

Here, we propose using a set of continuous observables \(\xi(x, y)\) and \(\zeta(u, z)\) that map uniquely to \(y\) and \(z\), respectively. For a state-inclusive controlled system, we have

\[
\begin{align*}
\psi_x(x_{k+1}, y_{k+1}) &= K_x \psi_x(x_k, y_k) + K_{xu} \psi_{xu}(x_k, y_k, u_k, z_k) \quad \text{(10)} \\
\psi_x(x, y) &= [x \xi(x, y) \varphi_x(x, y)] \quad \text{(11)} \\
\psi_{xu}(x, y, u, z) &= [u \zeta(u, z) \varphi_{xu}(x, y, u, z)], \quad \text{(12)}
\end{align*}
\]

where \(\varphi_x(x, y)\) and \(\varphi_{xu}(x, y, u, z)\) are observable functions (either fixed or learned). Functions such as

\[
\xi(x, y) = (2y - 1) \log \left(1 + e^{\rho(x, y)}\right) \quad \text{(13)}
\]

\[
\zeta(x, y) = \frac{1}{2} \tanh (\rho(x, y)) + y \quad \text{(14)}
\]

for binary and integer variables, respectively, provide a surjective mapping from \(\xi(x, y)\) to \(y\); \(\rho(x, y)\) is a function that can be fixed or learned just like \(\varphi_x(x, y)\). \[\[13\right)\]

\[\[14\right)\]

and \(\[\[14\right)\]\]

\[\[14\right)\]\]

are examples of possible options; others could be defined. Analogous definitions apply to \(\zeta(u, z)\). For both \(\xi(x, y)\) and \(\zeta(u, z)\), the surjective mapping makes it possible to eliminate the discrete variables without affecting the underlying dynamics. This in turn facilitates the use of data-driven computational methods to learn the KO and its observables.

C. Transformed Systems

Here, we introduce a different transformation between \(\[\[2\right)\]\]

and an equivalent system with continuous state variables; again, we retain the hybrid dynamics using only continuous variables. We then provide conditions for mapping the hybrid and continuous sets of trajectories to each other.

Theorem 2.1: Consider the system defined in \(\[\[2\right)\]\]

and \(\[\[3\right)\]\]

and define \(\sigma = [y z]\). Fixed control inputs \(\{u_k, z_k\}\) and initial conditions \((x_0, y_0)\) produces a trajectory \(\{x_k, y_k\}, k \in \mathbb{N}\). If

\[
\begin{align*}
f(x, y_1, u_1, z_1) &\neq f(x, y_2, u, z_2) \quad \text{(15)}
\end{align*}
\]

for all \(x\) and \(u\), and for all \(\sigma_1 \neq \sigma_2\), then there exists a dynamical system with continuous state variables \(s \in \mathbb{R}^{2n+m}\) and corresponding trajectory \(\{s_k\}\) such that there exists a bijection between \(s_k\) and \((x_k, y_k), k \in \mathbb{N}\).

Proof:

Let \(S \subseteq \mathbb{Z}^{p+q}\) be the set of possible \(\sigma\) values, and define \(\nu, \phi, \omega\) such that
The system has continuous state variables $x$ between initial conditions defined by a trajectory even in the presence of noise or other errors. Producing the surjective mapping using a mapping $x_{k+1} = z_k + \omega_k$.

Given $u_k$, we have a unique and bijective mapping between $(x_k, y_k)$ and $s_k = (x_k, u_k, v_k)$. The new dynamical system has continuous state variables $x$, $v$, and $u$ as well as control variables $\phi$ (continuous) and $\omega$ (discrete).

The proof of Theorem 2.1 uses the definition $v_k = x_{k+1} - x_k$. In this proof, $G(\cdot, x, u)$ maps from $V(x,u) \subset \mathbb{R}^n$, not $\mathbb{R}^n$, to $S$. However, it is possible to extend this. For example, using a mapping $V(\cdot, x, u) : \mathbb{R}^n \rightarrow V(x,u)$ such as

\[
V(v, x, u) = \arg \min_{v' \in V(x,u)} ||v' - v||
\]

produces the surjective mapping $G(V(\cdot, x, u), x, u) : \mathbb{R}^n \rightarrow S$; the mapping from $s_k$ to $(x_k, y_k)$ then becomes surjective. This extension is computationally useful because it makes the original hybrid trajectory recoverable from the transformed trajectory even in the presence of noise or other errors.

**Corollary 1.1:** Consider the uncontrolled hybrid system

\[
v_k = x_{k+1} - x_k
\]

\[
v_k = f(x_k, y_k, u_k, z_k) - x_k
\]

\[
u_{k+1} = u_k + \phi_k
\]

\[
z_{k+1} = z_k + \omega_k
\]

\[
v_{k+1} = f(x_{k+1}, y_{k+1}, u_{k+1}, z_{k+1}) - x_{k+1}
\]

\[
v_k + v_k, g(x_k, y_k, u_k, z_k), u_k + \phi_k, z_k + \omega_k
\]

\[
-x_k + v_k.
\]

If (15) holds, then there exists a family of bijections $G$.\[
\tilde{f}(x, u, \sigma_1) \neq \tilde{f}(x, u, \sigma_2) \forall x, u; \forall \sigma_1 \neq \sigma_2
\]

\[
V(x, u) = \left\{v : v = \tilde{f}(x, u, \sigma) - x \forall \sigma \in S\right\}
\]

\[
G(\cdot, x, u) : V(x, u) \rightarrow S
\]

\[
G(v, x, u) = \sigma : v = \tilde{f}(x, u, \sigma) - x
\]

\[
G(v_k, x_k, u_k) = \sigma_k
\]

\[
G_y(v_k, x_k, u_k) = y_k
\]

\[
G_z(v_k, x_k, u_k) = z_k
\]

\[
y_{k+1} = g(x_k, G_y(v_k, x_k, u_k), u_k, G_z(v_k, x_k, u_k))
\]

\[
v_{k+1} = f(x_k + v_k, h(x_k, u_k, v_k), u_k + \phi_k,
\]

\[
G_z(v_k, x_k, u_k) + \omega_k) - (x_k + v_k).
\]

The dynamics then consist of (16), (18), and (29) with initial conditions defined by $x_0$, $y_0$, $u_0$, $u_1$, $z_0$, and $z_1$.

\[
v_0 = f(x_0, y_0, u_0, z_0) - x_0
\]

\[
\phi_0 = u_1 - u_0
\]

\[
\omega_0 = z_1 - z_0
\]

Given $u_k$ and $z_k$, we have a unique and bijective mapping between $(x_k, y_k)$ and $s_k = (x_k, u_k, v_k)$. The new dynamical system has continuous state variables $x$, $v$, and $u$ as well as control variables $\phi$ (continuous) and $\omega$ (discrete).

where $x \in \mathbb{R}^n$ and $y \in \mathbb{Z}^p$. Fixed initial conditions $(x_0, y_0)$ produce a trajectory $\{x_k, y_k\}, k \in \mathbb{N}$.

\[
f(x, y_1) \neq f(x, y_2)
\]

for all $x$, and for all $y_1 \neq y_2$, then there exists a dynamical system with continuous state variables $s \in \mathbb{R}^n$ and corresponding trajectory $\{s_k\}$ such that there exists a bijection between $s_k$ and $(x_k, y_k), k \in \mathbb{N}$.

**Proof:** The result follows from Theorem 2.1 if $u_k$ and $z_k$ are set identically to zero.

To generate the necessary time series data, $v_k$ could be calculated after the simulations were run, and learning the KO would implicitly learn the mapping from $(x_k, y_k)$ to $v_k$. If (36) does not hold, though, then $v$ is not identifiable from $v$. Defining additional variables: $w_{k+1} = x_k$ and $w_{k+1} = v_k$ could circumvent this problem. For an uncontrolled system, the necessary condition would be

\[
f(f(x, y_1), g(x, y_1)) \neq f(f(x, y_2), g(x, y_2))
\]

or

\[
f(x, y_1) \neq f(x, y_2) \forall x, y_1 \neq y_2.
\]

(36) defines a single-step identifiability criterion. (37) defines a two-step version of that criterion: if the trajectories corresponding to different $y$ values differ after two time steps, then $y$ is identifiable by considering two successive data points from the same trajectory. In principle, essentially an $n$-step criterion (with associated additional variables) could be found, and there would be an analogous bijective mapping to the discrete variables. Additionally, the transformation described in the proof of Theorem 2.1 simplifies the functional forms of $K_x$ and $K_{zu}$, since the $x$ dynamics are known to be linear in $x$ and $v$ and the $u$ dynamics are linear in $u$ and $\phi$. The resulting transformed KO representation is

\[
\psi_x(x_{k+1}, u_{k+1}, v_{k+1}) = K_x \psi_x(x_k, u_k, v_k)
\]

\[
+ K_{zu} \psi_{zu}(x_k, u_k, v_k, \phi_k, \omega_k)
\]

(38)

\[
\psi_x(x, u, v) = [x \; u \; v \; \varphi_x(x, u, v)]
\]

(39)

\[
\psi_{zu}(x, u, v, \phi, \omega) = [\phi_k \; \omega_k \; \varphi_{zu}(x, u, v, \phi, \omega)]
\]

(40)

\[
K_x = \begin{bmatrix}
I_n & 0_{n \times m} & I_n & 0_{n \times n_z}
\end{bmatrix}
\]

(41)

\[
K_{zu} = \begin{bmatrix}
0_{m \times n} & I_m & 0_{m \times n_z} & 0_{m \times n_{zu}}
\end{bmatrix}
\]

(42)

The uncontrolled case has analogous structure. Using this transformation transfers the discontinuities from the state space into the dynamics, as does the general hybrid formulation described previously. In both cases, the motivation behind the transfer is computational. For learning the KO and its observables from data, and for using the KO to simulate trajectories, implicit hybrid or switching behaviour.
poses fewer difficulties than handling discrete state variables directly. This is particularly true for when using neural networks to learn the KO observables. Neural networks that produce discrete outputs cannot be trained using gradient-based learning algorithms because the outputs are effectively piecewise constant (i.e., their gradients are zero).

### III. Computational Demonstrations

#### A. Implementation

For testing, we used two computational examples: a simple numerical example and a vehicle automatic transmission. The neural networks used to learn the Koopman observables were implemented in Tensorflow [10]. These networks used Exponential Linear Unit (ELU) hidden layer activations, were trained to minimize relative Mean Squared Error (MSE)

\[ \sum_k \frac{\| \psi(x_{k+1}) - K \psi(x_k) \|^2}{\|x_{k+1} - x_k\|^2}, \quad (43) \]

and were trained using Adam [11] with an initial learning rate of 0.001. For the numerical test case, we used networks with four layers, each eight neurons wide (8x4) to represent \( \varphi \) with four layers, each eight neurons wide (8x4) to represent rate of 0.001. For the numerical test case, we used networks and were trained using Adam [11] with an initial learning

Sequential learning was sometimes beneficial in speeding up convergence, but generally, learning both simultaneously provided the greatest accuracy (with the warm start approximation as described for the general hybrid formulation).

#### B. Numerical Test Case

As an initial test case, we defined the hybrid system

\[
\begin{align*}
    x_{k+1} &= x_k + \frac{2y_k - 1}{1 + 0.1x_k^2} \\
    y_{k+1} &= \begin{cases} 
        1 & x_k < -1 \\
        0 & x_k > 1 \\
        y_k & \text{else}
    \end{cases}
\end{align*}
\]

(44)

For testing, we used two computational examples: a simple numerical example and a vehicle automatic transmission. The neural networks used to learn the Koopman observables were implemented in Tensorflow [10]. These networks used Exponential Linear Unit (ELU) hidden layer activations, were trained to minimize relative Mean Squared Error (MSE)

\[ \sum_k \frac{\| \psi(x_{k+1}) - K \psi(x_k) \|^2}{\|x_{k+1} - x_k\|^2}, \quad (43) \]

and were trained using Adam [11] with an initial learning rate of 0.001. For the numerical test case, we used networks with four layers, each eight neurons wide (8x4) to represent \( \varphi_x \), \( \varphi_{xu} \) and \( \rho \). The automatic transmission model, being more complicated, used a 12x4 network for each of \( \varphi_x \), \( \varphi_{xu} \) and \( \rho \); \( \varphi_x \) and \( \varphi_{xu} \) each had five elements for the numerical test case and ten elements for the automatic transmission.

The training data was generated from grids of initial points on the state and control variable spaces run for a single time step: a total of 1000 points for the numerical test case and 625,000 points for the automatic transmission model. The modified control variables in the transformed formulation changed the dimension of the space being sampled, so different sampling grids were used. This grid-based approach sampled the space uniformly without requiring ergodicity. We defined the grids such that the system dynamics would lie within the grid interior, which avoided boundary effects or a need for extrapolation. In the numerical example, mini-batch training sufficed. However, for the automatic transmission, full batch training was needed for good training convergence.

To generate trajectories from learned KO representations, we tested two different implementations. The first implementation (designated as ‘multiplied’) initialized \( \psi_x \) and propagated the dynamics by multiplying by \( K_x \) and adding \( K_{xu} \psi_{xu} \) (if necessary). State variable values were extracted in post-processing, but other than the initialization and calculation of \( \psi_{xu} \), \( x_k \) was not explicitly needed to calculate the trajectory. Control terms aside, this produced a purely linear representation of the dynamics. The second implementation (designated as ‘evaluated’) initialized and multiplied \( \psi_x \) as well, but at each iteration, it extracted state variable values and re-initialized \( \psi_x \). This implementation meant that the dynamics were not only a sequence of matrix multiplications, but it was intended to reduce error propagation across iterations and align with a rolling time horizon approach (such as MPC).

With the general hybrid formulation, we obtained better results when we learned a relaxed representation \( \xi_i = y_i \) to a lower level of fidelity and used that as a warm start to learn the representation using (14) as opposed to learning \( \rho \) and \( \varphi_x \) simultaneously. For the automatic transmission, there was also the question of whether to learn dynamics and control sequentially or whether to try and learn both simultaneously. Sequential learning was sometimes beneficial in speeding up convergence, but generally, learning both simultaneously provided the greatest accuracy (with the warm start approximation as described for the general hybrid formulation).

Fig. 1: Simulation Results (Formulation, Method, MSE; True trajectory. The trajectories produced by different learned Koopman representations; each caption’s MSE value, defined by (15), indicates the training error achieved by each method. These trajectory plots assess multi-step testing error; note the piecewise continuous nature of the true \( v \) trajectory. The switched representation was the easiest to train of the three methods, and it was also able to get the lowest training error;
the transformed formulation was able to train to an error of 0.005, and the general hybrid formulation using \([43]\) was able to train to 0.002. In all three cases, the evaluated method produced more accurate Koopman trajectories – trajectories that remained accurate over the 100 time steps presented in Fig. [1]. The multiplied method still produced good results in the switched and general hybrid formulations, and the worse results for the transformed formulation may be a reflection of its higher training error.

C. Vehicle Automatic Transmission

We used an automatic transmission model similar to those of Antsaklis and Koutsoukos [12] and of Lygeros et al. [13]:

\[
x_{k+1} = x_k + v_k \Delta t
\]

\[
v_{k+1} = v_k + \left[ -\alpha v_k \text{sign}(v_k) - g \sin(\theta(x_k)) \right]
\]

\[
-\gamma v_k G(q_k) + T_k \eta(v_k G(q_k)) \Delta t
\]

\[
q_{k+1} = \begin{cases} 
q_k + 1 & v_k G(q_k) > \omega_{\text{high}}, q_k < 5 \\
q_k - 1 & v_k G(q_k) < \omega_{\text{low}}, q_k < 1 \\
q_k & \text{else}
\end{cases}
\]

Here, \( -\alpha v_k \text{sign}(v_k) \) is deceleration due to air resistance, \( -g \sin(\theta(x_k)) \) is deceleration due to gravity on a sloping road, \(-\gamma v_k G(q_k)\) is deceleration due to engine friction, and \(T_k \eta(v_k G(q_k))\) is the acceleration due to torque provided by the engine; \(x_k\) is position, \(v_k\) is velocity, \(q_k\) is the gear (1 to 5), and \(G(q)\) is the gear ratio for each gear of the transmission. Fig. [2] shows trajectory results for the different formulations with a fixed \(\{T_k\}\) series and defined functions \(\theta(x)\) and \(G(q)\); again, the plots provide an assessment of multi-step testing error, and the listed MSE values indicate training error as defined by \([43]\). The switched formulation trained to MSE = 0.005 while the general hybrid formulation trained to 0.001 and the transformed formulation trained to 0.008. Again, the evaluated trajectories (blue) followed the true trajectory (green) more closely than the multiplied trajectories (orange). The evaluated position and velocity trajectories remained at least qualitatively accurate through for 40-50 time steps, whereas the multiplied trajectories were only good for 20-30 time steps. The discrepancies seemed worst when the Koopman trajectories produced \(q_k\) errors.

IV. DISCUSSION

Overall, the general hybrid formulation was the hardest to train. The switched formulation was the easiest to train and the most accurate, and the transformed formulation trained quickly but tended to plateau short of the desired accuracy. Larger neural networks might provide greater accuracy but would also require more effort to train. The general hybrid formulation may be the most scalable. As the number of discrete variables increases, the number of distinct states (and thus the number of different KO matrices to learn) for the switched formulation may grow exponentially. For the transformed formulation, recovering the discrete variable values and guaranteeing that \([15]\) or \([36]\) holds may become more difficult as \(p\) increases; this formulation’s additional variables also increase the sampling burden.

The neural networks used to learn the Koopman observables were small but required a relatively large amount of data to provide a sufficient level of accuracy. Mini-batch training can be a way to handle those large datasets, but training stability can suffer as a result. Strategic approaches may be useful here. For example, we found it advantageous to do initial training with smaller sets of data and then to use the larger, more time-consuming datasets after that pre-training had completed; this was like mesh refinement in finite element methods. Similarly, for the general hybrid formulation, doing pre-training with a simple continuous relaxation of the discrete variables \(y\) made it easier to get final convergence. These observations point towards larger questions of scalability and computational complexity for the methods presented. The current paper does not seek to address those questions in depth, but future work should consider those issues in the interest of making the methods more widely applicable.

Error propagation is a key factor in producing useful Koopman representations. Once training converged to the desired mean error level, the single-step error distributions were generally long-tailed. What may be even more important, though, are errors in the discrete transitions: significant trajectory deviations usually began with an inaccurate discrete variable transition from one state to another (or lack of such transition). Accurately predicting these transitions is of primary importance for long-term accuracy. One solution would be to change the training function. Weighting the training data so that \((x_k, x_{k+1})\) pairs representing discrete state transitions are given more importance would be one way to do this. Multi-step training would be another approach: it would replace \([43]\) with something like

\[
\min \sum_k \frac{||\psi_z(x_{k+2}) - K^2_z \psi_z(x_k)||^2}{||x_{k+2} - x_k||^2}. \quad (50)
\]

This would help to enforce accuracy across discrete transitions (e.g., at step \(k + 1\)). MSE may also not be the ideal metric for these problems. In the general hybrid formulation, for example, small MSE can lead to large trajectory errors because \(y\) is off by 1 even though the error on \(\xi\) is small.

V. CONCLUSIONS

We have produced and described three Koopman representations for hybrid systems. Two representations rely on surjective mappings to eliminate discrete variables. Initial demonstrations have provided some insight into how these methods could be scaled up to larger systems and applied in other contexts. One particular area for future work will be in modifying the training process to account for the specific challenges of hybrid systems; customized error functions (other than MSE) and multi-step training would fall under this heading. Another area will be comparing different discrete-to-continuous mapping functions for the general hybrid formulation in terms of trainability and accuracy.
Finally, it may be possible to produce additional hybrid Koopman formulations and compare them with the ones presented here.

VI. ACKNOWLEDGMENTS

The research described in this paper was funded by the National Security Directorate Seed Laboratory Directed Research and Development Investment at the Pacific Northwest National Laboratory, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy.

REFERENCES

[1] 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, 2015.

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

[3] E. Yeung, S. Kundu, and N. Hodash, “Learning deep neural network representations for koopman operators of nonlinear dynamical systems: Koopman operator meets model predictive control,” *Automatica*, vol. 93, pp. 149–160, 2018.

[4] C. Bakker, K. E. Nowak, and W. S. Rosenthal, “Learning koopman operators for systems with isolated critical points,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019.

[5] 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, p. e0150171, 2016.

[6] C. Bakker, W. S. Rosenthal, and K. E. Nowak, “Koopman representations of dynamic systems with control,” *arXiv preprint arXiv:1908.02233*, 2019.

[7] M. Budisic, R. Mohr, and I. Mezić, “Applied koopmanism,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, p. 047510, 2012.

[8] S. Hanke, S. Petiz, O. Wallemscheid, S. Klus, J. Böcker, and M. Dellnitz, “Koopman operator based finite-set model predictive control for electrical drives,” *arXiv preprint arXiv:1804.00854*, 2018.

[9] N. Govindarajan, H. Arbab, L. van Blargian, T. Matchen, E. Tegling, and I. Mezić, “An operator-theoretic viewpoint to non-smooth dynamical systems: Koopman analysis of a hybrid pendulum,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6477–6484, IEEE, 2016.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[11] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[12] P. J. Antsaklis, X. D. Koutsoukos, and N. Dame, “Hybrid systems control,” *Encyclopedia of Physical Science and Technology*, vol. 7, pp. 445–458, 2002.

[13] J. Lygeros, C. Tomlin, and S. Sastry, “Hybrid systems: Modeling, analysis and control,” tech. rep., University of California, Berkeley, 2008.