Deep Neural Networks With Koopman Operators for Modeling and Control of Autonomous Vehicles

Yongqian Xiao, Xinglong Zhang, Member, IEEE, Xin Xu, Senior Member, IEEE, Xueqing Liu, and Jiahang Liu

Abstract—Autonomous driving technologies have received notable attention in the past decades. In autonomous driving systems, identifying a precise dynamical model for motion control is nontrivial due to the strong nonlinearity and uncertainty in vehicle dynamics. Recent efforts have resorted to machine learning techniques for building vehicle dynamical models, but the generalization ability and interpretability of existing methods still need to be improved. In this paper, we propose a pure data-driven vehicle modeling approach based on deep neural networks with an interpretable Koopman operator. The main advantage of using the Koopman operator is to represent the nonlinear dynamics in a linear lifted feature space. In the proposed approach, a deep learning-based extended dynamic mode decomposition algorithm is presented to learn a finite-dimensional approximation of the Koopman operator. A multi-step prediction loss function is used in the training process, enabling a long-term prediction capability. Furthermore, a data-driven model predictive controller with the learned Koopman model is designed for velocity profile tracking control of autonomous vehicles. Simulation results in a high-fidelity CarSim environment show that our approach outperforms previously developed traditional and advanced modeling methods. Velocity profile tracking tests of the autonomous vehicle are also performed in the CarSim environment. The results show that our approach has better tracking accuracy and higher computational efficiency than the model predictive control algorithms using a nonlinear model and a linear time-varying model.

Index Terms—Vehicle dynamics, Koopman operator, deep learning, extended dynamic mode decomposition (EDMD), data-driven modeling, model predictive control (MPC).

I. INTRODUCTION

AUTONOMOUS vehicles are promising to free the hands of humans from tedious long-distance driving and have huge potential to reduce traffic congestion and accidents. A classic autonomous driving system usually consists of key modules of perception, localization, decision-making, trajectory planning, and control [1]–[3]. In trajectory planning and control, the information of vehicle dynamics is usually required for realizing agile and safe maneuvers, especially in complex and unstructured road scenarios [4].

In order to realize high-performance trajectory tracking of autonomous vehicles [5], [6], many control methods, such as Linear Quadratic Regulator (LQR) [2] and Model Predictive Control (MPC) [7], rely on vehicle model information with different levels and structures. However, obtaining a precise model of vehicle dynamics is difficult for the following reasons: i) the vehicle dynamics in the longitudinal and lateral directions are highly coupled; ii) strong nonlinearity and model uncertainty become dominant factors especially when the vehicle reaches the limit of tire-road friction [8]–[10]. Among the existing approaches, classic physics-based modeling methods rely on multiple physical parameters, which are required to be determined [11], [12]. Indeed, some model parameters, such as cornering stiffness coefficient, are not measurable and difficult to be estimated [11], [13], [14].

As an alternative to analytic modeling of vehicle dynamics, machine learning techniques have been studied in recent years for building data-driven models of vehicles [15], [16]. However, deep neural networks (DNN) commonly lack interpretability, which has recently been noted challenging for applications with safety requirements and remains a cutting-edge research topic. Also, due to the nonlinear activation functions, the obtained dynamic model is not easy to be used for designing a well-posed controller such as MPC, LQR, and so forth. Recently, the Koopman operator has been regarded as a powerful tool for capturing the intrinsic characteristics of a nonlinear system via linear evolution in the lifted observable space. To obtain a realistic model description, usually, dimensionality reduction methods such as Dynamic Mode Decomposition (DMD) in [17], [18] and Extended Dynamic Mode Decomposition (EDMD) in [19]–[21] can be used to approximate the Koopman operator with a finite dimension matrix. In [22], a data-driven identification method using the Koopman operator was proposed for vehicle dynamics but the basis functions were either manually designed or determined by the knowledge of the nonlinear dynamics, which is nontrivial for highly nonlinear dynamic systems. Also, previous works [22], [23] only consider a one-step prediction loss function in the identification process. The resulting model could be inaccurate in multi-step prediction.

To solve the above problems, this paper proposes a deep learning-based extended dynamic mode decomposition (Deep
EDMD) approach to approximate the Koopman operator of vehicle dynamics. In the proposed algorithm, lightweight deep neural networks serve as the encoder and decoder in the framework of EDMD for learning the Koopman operator. A multi-step prediction loss function is used in the training process, enabling long-term prediction capability. Furthermore, a data-driven MPC is designed for velocity profile tracking control of autonomous vehicles using the dynamic model learned by Deep EDMD. The simulation studies in a high-fidelity CarSim environment are conducted for performance validation.

The contributions of the paper are two-fold:

1) A deep learning-based EDMD approach i.e., Deep EDMD, is proposed for identifying an integrated vehicle dynamical model in both longitudinal and lateral directions. Different from other machine learning-based approaches, the resulting model is a linear time-invariant (LTI) model in the lifted space with a nonlinear mapping from the original state space. Extensive simulation results show that Deep EDMD has better modeling accuracy than the modeling methods with EDMD [22], a multi-layer perception (MLP) [24], long short-term memory (LSTM) [24], and the linear time-varying lateral dynamic model (LTV-LDM) [25].

2) A novel model predictive controller is designed with the learned model using Deep EDMD (called DE-MPC) for velocity profile tracking of autonomous vehicles. In DE-MPC, the linear part of the model is used as the predictor in the prediction horizon of the MPC. The velocity profile tracking results show that DE-MPC has better tracking accuracy and higher computational efficiency than MPC algorithms using a linear time-varying model and a nonlinear model (LTV-MPC [26] and NMPC [27]).

The rest of this paper is structured as follows. In Section II, the most related works are reviewed, while the dynamics of four-wheel vehicles and the preliminary on the Koopman operator using EDMD are shown in Section III. Section IV presents the main idea of the Deep EDMD method for data-driven model learning. Section V provides the design of a linear MPC with Deep EDMD for velocity profile tracking. The simulation results are reported in Section VI, while some conclusions are drawn in Section VII.

II. RELATED WORK

Earlier works in [28], [29] resorted to neural networks for the approximation of dynamics. In [30], a recurrent neural networks (RNNs)-based modeling approach was presented to model the dynamical steering behavior of an autonomous vehicle. Artificial neural networks (ANNs) with MLP were used in [15], [16] to model the vehicle dynamics in the longitudinal or lateral direction. Note that most of the above approaches built the vehicle dynamics in the sole longitudinal direction or lateral direction. Identifying an accurate coupled vehicle dynamics model is challenging due to the high nonlinearity.

The Koopman operator has been recently realized to be a powerful tool for representing nonlinear dynamics. The main idea of the Koopman operator was initially introduced in [31], with the goal of describing the nonlinear dynamics using a linear model with the state space constructed by observable functions. It has been proven that the linear model with the Koopman operator can ideally capture all the characteristics of the nonlinear system as long as the lifted state space adopted is invariant. That is to say, the structure with the Koopman operator is interpretable. To obtain a finite-dimensional approximation of the Koopman operator, DMD [17], [32] and EDMD [19], [33], [34] can be used. Unlike the observables are physically meaningful in DMD, the observable functions in EDMD are constructed using basis functions, e.g., Gaussian functions, polynomial functions. However, the approximation performance might be sensitive to basis functions. Hence, the selection of basis functions requires specialist experience.

Motivated by the above problems, recent works have resorted to designing deep neural networks to generate observable functions automatically. In [35], an autoencoder is utilized to learn dictionary functions for unforced dynamics. The finite-dimensional approximation of the Koopman operator was computed via solving a least-squares problem. In [36], a deep variational Koopman (DVK) model was proposed with an LSTM network for learning the distribution of the observable functions. A deep learning network was used in [37] to learn the Koopman operator of nonlinear systems with continuous spectra, and it was extended to design an optimal controller with a linear controller design methodology [38]. From the application aspect, a deep DMD method has been proposed for the modal analysis of fluid flows in [39]. In [40], [41], the Koopman operator was used for the modeling of power systems, and its extension with deep learning-based feature representation was developed in [42] to improve the modeling performance. Another recent work utilizes the transformer to enhance the feature representation ability of the Koopman operator for physical systems identification [43]. In [44], a highway traffic prediction model was built based on the Koopman operator. Other applications can be found for representing neurodynamics in [45], molecular dynamics in [46], [47], and model reduction in [48]. In the aspect of robot modeling and control, a DMD-based modeling and controlling approach was proposed for soft robotics [49].

In modeling of autonomous vehicle dynamics, some prior approaches based on the Koopman operator using EDMD were proposed in [22], [50], [51] for modeling and control of vehicle dynamics, where the kernel centers of the kernel-based observable functions were manually selected. In [23], the Koopman operator with MLP was utilized to realize data-driven optimal control of a small-size car. The weights of the neural network were trained according to the gradient-based updating rule, while the model parameters in the lifted space were calculated separately by least-squares methods using a one-step model prediction loss function. The resulting model could be inaccurate in multi-step prediction. Motivated by the above issues, we propose a deep neural network to learn the observable functions of the Koopman operator in the framework of EDMD, with the goal of data-driven modeling of vehicles in both longitudinal and lateral directions. The deep neural networks serve as the encoder and decoder whose weights are automatically updated in the learning process, avoiding the manual selection problem of observable functions. Moreover, a multi-step prediction loss
function is constructed and minimized to improve the long-term prediction accuracy. The main difference and advantages of our approach to related works [22], [23], [50], [51] are summarized in Table I.

### III. VEHICLE DYNAMICS MODEL AND EDMD FOR KOOPMAN OPERATOR

In this section, the basic description of the vehicle dynamics is given, and the preliminary on the Koopman operator using EDMD is introduced.

#### A. The Basic Model Description of Autonomous Vehicles

In this paper, we adopt the type of autonomous vehicles consisting of four wheels with front-wheel-steering functionality. A sketch of the vehicle dynamics is depicted in Fig. 1. Denote \( x = [v_x, v_y, \psi]^\top \) and \( u = [\zeta, \eta]^\top \in \mathcal{U} \) as the state and control input of the vehicle dynamics respectively, where \( v_x, v_y \), and \( \psi \) are the longitudinal and lateral velocities, and the yaw rate; \( \zeta \) and \( \eta \) are steering wheel angle and the engine, \( \mathcal{U} \) is a convex set. The mathematical expression of the vehicle dynamics is described as follows:

\[
\dot{x} = f_c(x, u) \tag{1}
\]

where \( x \in \mathbb{R}^n \) and \( u \in \mathbb{R}^m \) are the state and control input, \( n = 3, m = 2 \), the mapping \( f_c : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \) is unknown. As noted in [8], [52] (see Fig. 1), the vehicle dynamics show strong couplings in the longitudinal and lateral directions and strong nonlinearities due to the tire characteristics. Some prior simplified vehicle dynamical models [22], [50], [51] were adopted to reduce the model nonlinearity. Even so, identifying the internal crucial cornering stiffness parameters of the tires is still challenging.

In this paper, we propose a deep neural network method based on the Koopman operator to learn the vehicle dynamics of (1) at a wide operating range. We do not assume any knowledge of the internal parameters of the vehicle dynamics. Only pre-collected data sets on states and control inputs are utilized for training the weights of the deep neural network (DNN) to obtain a precise representation of (1).

The resulting model consists of a static nonlinear DNN to map the state space to a lifted one and a linear dynamic model evolution in the lifted state space, see Section IV. Such a feature allows us to design a linear standard model predictive control algorithm for high-performance tracking control of vehicle dynamics. The MPC design with the Koopman model is deferred in Section V.

#### B. The Koopman Operator Using EDMD

The Koopman operator was commonly used to capture the intrinsic characteristics via a linear dynamical evolution for unforced nonlinear dynamics [17], [19]. With a slight change, the Koopman operator can also be used for representing systems with control inputs. Let

\[
x_{k+1} = f(x_k, u_k) \tag{2}
\]

be the discrete-time version of (1) with a specified sampling interval, where \( k \) is the discrete-time index. According to [33], the generalization of the Koopman operator for model (2) relies on an extended state variable \( z = [x^\top, u^\top]^\top \), where \( u := u_0^\infty \) denotes all the controls in the control space \( \mathcal{U} \). In line with [33], the Koopman operator on (2) with the extended state \( z \) is described as

\[
K \varphi(z_k) = \varphi(z_{k+1}), \tag{3}
\]

where \( K \) denotes the Koopman operator, which is infinitely dimensional; \( \varphi(z_k) \in \mathbb{R} \) is the observable in the lifted space. Please refer to [33] for more details on the definition of the Koopman operator.

A finite-dimensional approximation of \( K \) is of interest for controller design. As the state \( z_k \) is of infinite dimension, we
adopt \( \varphi(z_k) = [\varphi_1(x_k)^T \cdots \varphi_L(x_k)^T \ u_k^T]^T \in \mathbb{R}^{L+m} \) as a group of observable functions for the practical calculation, where \( L \) is the number of observable functions on \( x \), i.e., \( \varphi_i(x) \). The main idea of computing the Koopman approximation with EDMD consists of two steps. Firstly, select the observable functions as basis functions, e.g., radial basis functions (RBF) with different kernel centers and widths. Secondly, compute a finite-dimensional approximation of the Koopman operator by the least-squares method. To this end, for forced dynamics \( (1) \), we define \([A \ B] \) as the first \( L \) rows of the approximated Koopman operator, where \( A \in \mathbb{R}^{L \times L} \) and \( B \in \mathbb{R}^{L \times m} \) are the system matrices of the lifted model. Therefore, the resulting Koopman model in the observable space can be written as

\[
\varphi(x_{k+1}) = [A \ B] \begin{bmatrix} \varphi(x_k) \\ u_k \end{bmatrix},
\]

(4)

Matrices \( A \) and \( B \) can be computed via solving a least-squares problem based on a data set with \( M \) snapshots, i.e., \( \{x_i, u_i, x^+_{i|M}\}_{i=1}^M \), where \( x^+_{i|M} \) is the evolution of \( x_i \) with \( u_i \). Consequently, the optimization problem can be described as

\[
\min_{A, B} \sum_{k=1}^M \|\varphi(x^+_{i|M}) - [A \ B] \begin{bmatrix} \varphi(x_i) \\ u_i \end{bmatrix} \|^2_F.
\]

(5)

By solving (5), the analytical solution of \([A \ B] \) is given as

\[
[A \ B] = VW^\top (WV^\top)^{-1},
\]

(6)

where

\[
W = \begin{bmatrix} \varphi(x_{[1]}) & \cdots & \varphi(x_{[M]}) \\ u_{[1]} & \cdots & u_{[M]} \end{bmatrix},
\]

\[
V = [\varphi(x_{[1]}) \cdots \varphi^+(x^+_{[M]})].
\]

Let the matrix \( C \in \mathbb{R}^{n \times L} \) be defined to project \( \varphi(x) \) to the original state \( x \). Similarly, we can obtain the matrix \( C \) by solving

\[
\min_C \sum_{k=1}^M \|x_k - C\varphi(x_k)\|^2_F,
\]

(7)

leading to

\[
C = X\Xi^\top (\Xi\Xi^\top)^{-1},
\]

(8)

where \( X = [x_{[1]} \cdots x_{[M]}], \Xi = [\varphi(x_{[1]}) \cdots \varphi(x_{[M]})] \).

IV. THE DEEP EDMD APPROACH FOR VEHICLE DYNAMICS MODELING

In this section, we present the main idea of the Deep EDMD algorithm for modeling the vehicle dynamics, where a DNN is utilized to construct an observable subspace of the Koopman operator automatically.

Different from EDMD, we propose a Deep EDMD algorithm for modeling the vehicle dynamics in which a DNN is utilized to construct an observable subspace of the Koopman operator automatically. Also, the merit of Deep EDMD with respect to classic MLP is that, the deep learning network is integrated into the EDMD method to learn an estimate of the Koopman operator with finite dimension. To proceed, one can write the approximated dynamics resulting from Deep EDMD in the following form:

\[
\begin{aligned}
\phi_e(x_t, \theta_e, u_t) \\
\dot{x}_t = \tilde{\Psi} (\phi_e(x_t, \theta_d),
\end{aligned}
\]

(9)

where \( K = [A \ B] \in \mathbb{R}^{L \times N} \), and \( N = L + m \). \( \phi_e(x_t, \theta_e) \) is the encoder parameterized with weights \( \theta_e \), \( \Psi(x_t, \theta_d, u_t) \) is \( \phi_e(x_t, \theta_d)^\top u_t \) \( \in \mathbb{R}^N \), and \( \tilde{\Psi}(\cdot) \) denotes the decoder parameterized with weights \( \theta_d \). Without abuse of notations, we use \( \Psi \) to stand for \( \Psi(x_t, \theta_d, u_t) \) unless otherwise specified in the rest of the paper.

As shown in Fig. 2, the Deep EDMD algorithm relies on an autoencoder structure. The encoder, i.e., \( \phi_e \), including a neural network constructed by fully-connected layers, is in charge of mapping the original state to a lifted observable space. The weights \( A \) and \( B \) are placed connecting to the last layer of the encoder without activation functions. In principle, \( A \) and \( B \) can be trained synchronously with the encoder. In case the encoder might experience unanticipated errors and vanishing gradient, \( A \) and \( B \) can be updated as the last training step in which way, the worst scenario of Deep EDMD might degrade into EDMD. Similar to the encoder, the decoder consists of fully-connected layers, devotion to recovering the original state from the lifted observable space. To be specific, at any hidden layer \( l \), the output can be described as

\[
\hat{y}_l(e) = \sigma_l(e) \left( W_e(l)e_l^{(l-1)} + b_l(e) \right)
\]

(10)

where \( * = e, d \) in turn stands for the subscripts for the encoder or decoder, \( W_e(l) \in \mathbb{R}^{n_l \times n_{l-1}} \) and \( b_l(e) \in \mathbb{R}^{n_l} \) are the weight and bias of the hidden layer \( l \), where \( n_l \) represents the number of neurons of the hidden layer \( l \), \( \sigma_l(e) \) denotes the activation function of the hidden layer \( l, l = 1, \ldots, H \), where \( H \) is the number of hidden layers of the encoder and decoder. In the encoder, \( \hat{y}_e(l) = x_t, \sigma_l(e), l = 1, \ldots, H - 1 \) is designed using a rectified linear unit (ReLU) [53], while no activation function is used in the last layer. As for the decoder, \( \sigma_l(d), l = 1, \ldots, H - 1 \) uses ReLU as the activation function, while the last layer adopts a Sigmoid activation function. The lifted state can be obtained by the encoder, i.e.,

\[
\phi_e(x_t, \theta_e) = [x_t^\top \ (\hat{y}_e^{(H)}(e)]^\top,
\]

(11)

where \( \hat{y}_e^{(H)}(e) \in \mathbb{R}^{k-n} \) is the output of the last layer of the neural network in the encoder, which can be computed with (10) based on \( x_t \).

Similarly, the reconstructed state can be computed by the decoder based on the lifted state:

\[
\hat{x}_t = \tilde{\Psi} (\phi(e, \theta_d)).
\]

(12)

Note that the input of the decoder is the lifted state instead of the output of the neural network in the encoder, i.e., \( \hat{y}_e^{(H)}(e) = \phi_e(x_t) \). Concerning (10), one can promptly get the whole expressions of the encoder from (11) and the decoder from (12).

As the objective is to approximate the vehicle dynamics in a long time window, the multi-step prediction error instead of the one-step one is to be minimized. Hence the state and control
sequences with time information are used to formulate the optimization problem. Different from that in EDMD with one-step prediction approximation, the resulting problem is difficult to be solved analytically, but it can be trained in a data-driven manner. To introduce the multi-step prediction loss function, we first write the state prediction in time steps, which leads to the loss function being defined as

\[ L = \sum_{i=1}^{p} \left\| x_{t+i} - \hat{x}_{t+i} \right\|^2. \]  

where \( r_{x,p} \in \mathbb{R}^n \) is the prediction error at the \( p \)-th step, and \( K^{[p]} \Psi_t \) is the \( p \)-step ahead state starting from \( x_t \), i.e.,

\[
K^{[p]} \Psi_t = \phi_e(x_{t+p}, \theta_e)
\]

\[
= A^p \phi_e(x_{t+p-1}, \theta_e) + ABu_{t+p-1}
\]

\[
= A^p \phi_e(x_{t}, \theta_e) + \sum_{i=1}^{p} A^{i-1} Bu_{t+p-i}
\]

Specifically, we minimize the sum of prediction errors along the \( p \) time steps, which leads to the loss function being defined as

\[ \mathcal{L}_{x,x} = \frac{1}{p} \sum_{i=1}^{p} \left\| x_{t+i} - \hat{x}_{t+i} \right\|^2. \]  

Another perspective of validating the modeling capability of Deep EDMD is to evaluate the prediction error in the lifted observable space, i.e., it is crucial to minimize the error of the state evolution in the lifted space and the lifted state sequence mapping from the real dynamics. To this objective, we adopt the loss function in the lifted linear space as

\[ \mathcal{L}_{x,o} = \frac{1}{p} \sum_{i=1}^{p} \left\| \phi_e(x_{t+i}, \theta_e) - K^{[i]} \Psi_t \right\|^2. \]  

In order to minimize the reconstruction error, the following loss function about the decoder is to be minimized:

\[ \mathcal{L}_{o,x} = \frac{1}{p} \sum_{i=1}^{p} \left\| x_{t+i} - \Psi(\phi_e(x_{t+i}, \theta_e), \theta_d) \right\|^2. \]  

Also, to guarantee the robustness of the proposed algorithm, the loss function in the infinite norm is adopted, i.e.,

\[ \mathcal{L}_{\infty} = \frac{1}{p} \sum_{i=1}^{p} \left\| x_{t+i} - \Psi(\phi_e(x_{t+i}, \theta_e), \theta_d) \right\|_{\infty}. \]

With the above loss functions introduced, the resulting learning algorithm aims to solve the following optimization problem:

\[ \min_{\theta_e, \theta_d, \theta, A, B} \mathcal{L}, \]

where \( \mathcal{L} \) is the overall optimization function defined as

\[ \mathcal{L} = \alpha_1 \mathcal{L}_{o,x} + \alpha_2 \mathcal{L}_{x,x} + \alpha_3 \mathcal{L}_{x,o} + \alpha_4 \mathcal{L}_{\infty} \]

\[ + \alpha_5 \| \theta_e \|^2 + \alpha_6 \| \theta_d \|^2, \]

where \( \alpha_i, i = 1, \ldots, 6 \), are the weighting scalars, \( \| \theta_e \|^2, \| \theta_d \|^2 \) are the regularization term used for avoiding over-fitting.

Since the least-squares method cannot handle nonlinear terms derived from multi-step loss functions, i.e., \( K^{[p]} \), and backpropagation of corresponding gradients to the autoencoder, an Adam [54] solver using gradient descent is adopted in this paper, see Algorithm 1. After training, the resulting approximated dynamics for (1) can be given as

\[
\begin{aligned}
\phi_e(x_{t+i}, \theta_e) &= A\phi_e(x_t, \theta_e) + Bu_t \\
\hat{x}_{t+i} &= \Psi(\phi_e(x_{t}, \theta_e), \theta_d).
\end{aligned}
\]
Algorithm 1: Deep EDMD for Vehicle Dynamics Modeling.

Require: Initialize $\theta_c, \theta_d, A, B, p, Epoch = 0, Epoch_{\max}, \alpha_i, i = 1, \ldots, 6$, batch size $b_c$, a small scalar $\epsilon > 0$.
Ensure: trained $\theta_c, \theta_d, A, B$;
1: while Epoch $< Epoch_{\max}$ or $|\mathcal{L}| > \epsilon$ do
2: Reset the training episodes;
3: while Epoch is not Terminated do
4: Sample a batch data sequence of state and control inputs, i.e., $X = \{x^{[i]}_{0:p}\}_{i=1}^b, U = \{u^{[i]}_{0:p-1}\}_{i=1}^b$;
5: Obtain lifted states $\phi_c(x^{[i]}_{1:p}, \theta_c)$ with (11) and reconstructed states $x^{[i]}_1 = \hat{\Psi}(\phi_c(x^{[i]}_{1:p}, \theta_c), \theta_d)$ with (12), for all $i = 0, \ldots, p$ and $j = 1, \ldots, b$;
6: Compute multi-step lifted states $K^{[i]}\Psi_0$ with (14) and predicted states $\hat{\Psi}(K^{[i]}\Psi_0)$, where $i = 1, 2, \ldots, p$;
7: Obtain the weighted loss $\mathcal{L}$ with (20) based on (15)–(18);
8: Update $\theta_c, \theta_d, A, B$ and $\bar{B}$ via solving (19) with an Adam optimizer [54];
9: end while
10: Epoch = Epoch + 1
11: end while

V. MPC WITH DEEP EDMD FOR VELOCITY PROFILE TRACKING OF AUTONOMOUS VEHICLES

In this section, a novel model predictive controller based on Deep EDMD, i.e., DE-MPC, is proposed to show how the learned dynamical model can be utilized for controlling the vehicle in a velocity profile tracking problem. To this end, the learned model (21) is firstly used to define an augmented version with the input treated as the extended state, i.e.,

$$\begin{cases}
\xi_{t+1} = \tilde{A}\xi_t + \tilde{B}\Delta u_t \\
y_t = C\xi_t,
\end{cases}$$

subject to:
1) model constraint (22) with initialization (23);
2) the constraints on control and its increment:

$$u_{\text{min}} \leq u_t \leq u_{\text{max}}$$

$$\Delta u_{\text{min}} - \epsilon \mathbf{1}_m \leq \Delta u_t \leq \Delta u_{\text{max}} + \epsilon \mathbf{1}_m,$$

where $N_{p}, N_{c}$ are the prediction and control horizons, respectively, $Q, R \in \mathbb{R}^{L \times L}, \mathbb{R}^{m \times m}$ are positive-definite matrices for penalizing the tracking errors and the control increment, while $\rho > 0$ is the penalty of the slack variable $\epsilon$, and $\mathbf{1}_m \in \mathbb{R}^m$ is a vector with all the entries being 1. Note that we use $N_{p} \geq N_{c}$ and the control increment $\Delta u_t = 0$ is assumed for all $t, N_{c} < t \leq N_{p}$. $u_{\text{min}}, u_{\text{max}}, \Delta u_{\text{min}}$ and $\Delta u_{\text{max}}$ are the minimum and maximum values of the control and its increment. In the following we will reformulate (24a) in a more straightforward manner. First, let $
abla_t = [y_{t+1} y_{t+2} \cdots y_{t+N_{c}}]^\top$, one can compute

$$\gamma_t = \Gamma \xi_t + \Theta \Delta u_t,$$

where $\Gamma = \begin{bmatrix} C\bar{A} & CA^2 & \cdots & CA^{N_{p}} \end{bmatrix}^\top$,

$$\Theta = \begin{bmatrix} C \bar{B} & 0 & \cdots & 0 \\
CA\bar{B} & C\bar{B} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
CA^{N_{p} - 2}\bar{B} & CA^{N_{p} - 3}\bar{B} & \cdots & CA^{N_{p} - N_{c}}\bar{B} \end{bmatrix},$$

$G_t = [g_{t+1}^\top g_{t+2}^\top \cdots g_{t+N_{c}}^\top]^\top$, for $G = U, g = u$ and $G = \Delta U, g = \Delta u$. Therefore, we can reformulate optimization problem (24) as

$$\min_{\Delta U_{t+\epsilon}} J(\xi_t, \Delta U_t, \epsilon) = \begin{bmatrix} (\Delta U_t^\top \epsilon) \end{bmatrix}^\top \mathcal{H} [\Delta U_t^\top \epsilon]^\top + G_t \begin{bmatrix} (\Delta U_t^\top \epsilon) \end{bmatrix}^\top + \mathcal{P}t$$

s.t. $\Delta U_{\text{min}} - \epsilon \mathbf{1}_{mN_{c}} \leq \Delta U_t \leq \Delta U_{\text{max}} + \epsilon \mathbf{1}_{mN_{c}}$

$$U_{\text{min}} \leq U_t \leq U_{\text{max}}$$

$$\epsilon \geq 0$$

(26)

where $\mathcal{H} = \text{diag}(\Theta^\top Q_y \Theta + R_y, \rho)$, $G_t = [2E_t^\top Q_y \Theta 0]$, $\mathcal{P}t = E_t^\top Q_y E_t$ and $E_t = \Gamma \xi_t - \gamma_{t+1, \epsilon}$, and $\gamma_{t+1, \epsilon}$ is defined as the sequence $y_{t+1, \epsilon}, y_{t+2, \epsilon}, \ldots, y_{t+N_{c}, \epsilon}$. The matrices $Q_y = \text{diag}(Q, Q, \ldots, Q)$ in $\mathbb{R}^{K N_{c} \times K N_{c}}$ and $R_y = \text{diag}([R, \ldots, R]) \in \mathbb{R}^{mN_{c} \times mN_{c}}$.

Let $\Delta U_{t+\epsilon} = [\Delta U_{t+\epsilon}^{[1]} \cdots \Delta U_{t+\epsilon}^{[t]} N_{c}-1]^{\top}$ be the optimal solution to (26) at time $t$, the control applied to the system is

$$u_{t|t} = u_{t-1} + \Delta u_{t|t}. \quad (27)$$

Then the optimization problem is solved again in the time $t + 1$ according to the moving horizon strategy. To clearly illustrate the DE-MPC algorithm, the implementing steps are summarized in Algorithm 2.

Remark 1: The proposed approach results in a Koopman model, consisting of a static nonlinear encoder network and a linear time-invariant evolution in the lifted state space. Consequently, one can formulate an MPC problem using the linear...
Algorithm 2: DE-MPC for Velocity Profile Tracking.

**Require** the trained $\theta_e$, $\theta_d$, A, B; prediction and control horizon $N_p$, $N_c$; weights matrices $Q$, $R$, $\rho$;

1: for $i = 1, 2, \cdots$ do
2: Get the current lifted state $\mathbf{x}_t := \phi_t(x_t, \theta_e)$ and reference $y_{ref,t}$ using the planned trajectory according to (11);
3: Solve (26) to compute $\Delta u_{t|t}$;
4: Compute $u_{t|t}$ with (27) and apply it to the vehicle dynamical system.
5: end for

control design. Hence, the overall computational complexity is mainly due to two parts. The first is due to the inference of the static encoder network. We roughly assume the encoder with $H + 1$ layers has the same number of neurons $\ell$ of each layer. Then the corresponding computational complexity of the encoder is $O(H\ell^3)$. The second part is due to the standard MPC, whose computational complexity is roughly $O(N_p^3 L^3)$ by an interior point method and is reduced to $O(N_p L^3)$ if a block-diagonal structure is used [55], where $L$ is the number of observable functions.

VI. SIMULATION AND PERFORMANCE EVALUATION

In this section, the proposed Deep EDMD and DE-MPC methods were validated in a high-fidelity CarSim simulation environment. Deep EDMD was compared with modeling methods using EDMD [22], MLP [24], LSTM [24], and LTV-LDM [24] to show its effectiveness. Furthermore, DE-MPC was compared with two MPC algorithms, i.e., nonlinear MPC (NMPC) and linear time-varying MPC (LTV-MPC), to demonstrate the advantages of our approach in computational efficiency and tracking performance for velocity profile tracking.

A. Data Collection and Preprocessing

We validated our approach in a high-fidelity CarSim environment in which the dynamics were modeled with real vehicle data sets. As shown in Fig. 3, an F-Class sedan car with a 300 KW engine in CarSim 2019 was chosen as the original Car model. Combining Simulink/MATLAB 2017b, input-output data sets were collected with a Logitech driving hardware controlled by a human being on a flat square. The collected data sets were normalized according to (11) and (32). We collected about $7.7 \cdot 10^5$ snapshots of data consisting of 40 episodes and randomly chose 2 episodes as the testing sets, 2 episodes for validation, and the rest 36 episodes were regarded as the training sets. To enrich the diversity of the training data, we randomly generated the sampling starting point in the range of $[0, p]$ so that we could sample data sequences from a different starting position before each training epoch.

B. Parameter Settings and Training of All Comparative Methods

**Deep EDMD:** In Deep EDMD, the structure of the encoder was of five layers with the structure chosen as $[n ~ 32 ~ 64 ~ L ~ n]$. And the structure of the decoder was set as $[L ~ 128 ~ 64 ~ 32 ~ n]$. The simulations were performed with $L$ being set as 13. All the parameters used in the simulation are listed in Table II. The collected data sets were normalized according to $x = (x - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}})$. We collected about $7.7 \cdot 10^5$ snapshots of data consisting of 40 episodes and randomly chose 2 episodes as the testing sets, 2 episodes for validation, and the rest 36 episodes were regarded as the training sets. To enrich the diversity of the training data, we randomly generated the sampling starting point in the range of $[0, p]$ so that we could sample data sequences from a different starting position before each training epoch.

| Hyperparameter | Value | Hyperparameter | Value |
|----------------|-------|----------------|-------|
| Learning rate  | $10^{-3}$ | Batch size     | 64    |
| $L$            | 13    | $\alpha_2$    | 1.0   |
| $\alpha_1$    | 1.0   | $\alpha_4$    | $10^{-9}$ |
| $\alpha_3$    | 0.3   | $\alpha_6$    | $10^{-9}$ |
| $\beta_1$     | 1.0   | $\beta_2$     | $10^{-9}$ |
| $\beta_3$     | $10^{-9}$ |

The collected data sets were normalized according to $x = (x - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}})$. Recall that the value of the steering wheel angle was limited in the range of $[−450^\circ, 450^\circ]$. The collected data sets were normalized according to $x = (x - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}})$. We collected about $7.7 \cdot 10^5$ snapshots of data consisting of 40 episodes and randomly chose 2 episodes as the testing sets, 2 episodes for validation, and the remaining 36 episodes were regarded as the training sets. To enrich the diversity of the training data, we randomly generated the sampling starting point in the range of $[0, 1]$ so that we could sample data sequences from a different starting position before each training epoch.

1The data sets and the CarSim parameter file of the vehicle can be downloaded: https://drive.google.com/drive/folders/1wqPPr6-Ck7AUzSXEkAtAh68bO3O3QI?usp=sharing or https://pan.baidu.com/s/15aEUnMB9gHUKh8eVEKCDzw?pwd=bni3h
randomly with a normal distribution $\mathcal{N}(\mu, \sigma^2)$

$$\varphi(x) = \|x - \kappa\|^2 \log(\|x - \kappa\|),$$

(28)

where $\sigma$ and $\mu$ are the standard deviation and mean of the data sets.

**MLP [24]:** Also, an MLP was used for comparison, to show the strength of our approach in terms of modeling and robustness. It has a similar structure to [24] but has different states and control inputs. The structure of the MLP method was chosen as $[n + m \ 32 \ 64 \ 128 \ 128 \ 64 \ 32 \ n]$. Two scenarios were considered in the comparison. In the first scenario, all the parameters were regarded as optimization variables to be learned, while in the second scenario, an additional hidden layer was added in Deep EDMD and MLP, respectively, where weights and biases were updated randomly with a normal distribution during the whole training process. The weights and biases of MLP were parameterized by $\theta_M$, and the loss function of MLP was given as

$$L_{mlp} = \frac{1}{p} \sum_{i=1}^{p} \|x_i - \hat{x}_i\|^2_2 + \frac{1}{p} \sum_{i=1}^{p} \|\theta_M\|^2_2,$$

(29)

where $\hat{x}$ denotes the one-step ahead prediction with $\hat{x}_{i-1}$, and $x_0 = x_0, \beta_1, \beta_2,$ and $\beta_3$ are weighting scalars for each term of the loss function, and their values are given in Table II.

The first term denotes the multi-step predicted loss. The second term is the infinite norm to penalize the largest error of the prediction, and the last term is the $l_2$ regularization term used for avoiding over-fitting.

**LSTM [24]:** LSTM is a classic method to cope with sequence-to-sequence (S2S) prediction problems, and was used to predict the acceleration and angular velocity of vehicles [24]. In this work, LSTM recursively predicts the next state by taking the last predicted state and current control inputs as the input. The structure of LSTM contains three LSTM cells, where each LSTM cell has the same input and output dimension and equal $n$. The second and third LSTM cells take the outputs of the first and second ones as inputs, respectively. The last LSTM cell is connected with an MLP with the structure designed as $[n \ 128 \ 128 \ n]$ to predict the next state. The three LSTM cells adopt the vanilla LSTM cell [57], which consists of a cell, a forget gate, an input gate, and an output gate. Each layer of the MLP in LSTM has a ReLU as the activation function, except the last layer is linear. In training, the designed LSTM and MLP methods had the same loss function in (29) and hyper-parameters in Table II, such as the step $p$ and the learning rate.

**LTV-LDM [25]:** Except for those above mentioned data-driven approaches, an analytic LTV-LDM [25] of the vehicle was also considered as a comparison. LTV-LDM predicts the lateral velocity and yaw rate based on the true longitudinal velocity. Unlike data-driven approaches, the prediction accuracy of LTV-LDM relies on exact internal parameters of vehicle dynamics, e.g., cornering stiff coefficients and vehicle inertia. These parameters are difficult to be estimated precisely in practice and were assigned by values from CarSim in this work.

### Table III

| Testing case | $v_x$ (m/s) | $v_y$ (m/s) | $\psi$ (rad/s) |
|--------------|-------------|-------------|----------------|
| 1            | 0.75        | 1.99        | 1.78           |
| 2            | 0.04        | 0.36        | 0.14           |
|              | 0.02        | 0.20        | 0.08           |

| Testing case | $v_x$ (m/s) | $v_y$ (m/s) | $\psi$ (rad/s) |
|--------------|-------------|-------------|----------------|
| 1            | 1.15        | 1.42        | 2.80           |
| 2            | 0.08        | 0.25        | 0.16           |
|              | 0.05        | 0.66        | 0.12           |

Since Deep EDMD, MLP, and LSTM utilize neural networks, they were trained using Python API in TensorFlow [58] framework with an Adam solver [54], and using an NVIDIA GeForce GTX 2080 Ti GPU. EDMD is based on least-squares methods and was trained in MATLAB with an Intel i9-9900K@3.6 GHz.

In this paper, the neural networks of Deep EDMD, MLP, and LSTM have $2.54 \times 10^4, 2.7 \times 10^4$, and $1.77 \times 10^4$ training weights, respectively. Due to the usage of deep neural networks, Deep EDMD, MLP, and LSTM need a longer training time (several hours) but have much higher modeling precision. EDMD needs shorter training time (approximately 5~10 seconds) since it can be solved by the least-squares method, but at the cost of lower modeling precision due to the manual design of observable functions. As for control methods, they were all implemented in the same CarSim/MATLAB environment with the same computer. DE-MPC and LTV-MPC were implemented using the quadprog solver, while NMPC was optimized by the fmincon solver.

### C. Performance Evaluation

**Validation of the learned model with Deep EDMD:** The trained models with all the algorithms were validated with the same training data set. The resulting state predictions of all the algorithms under the same control profile are illustrated in Fig. 4. The true value trajectories in Fig. 4 were sampled from the testing data set and are chosen to get a wide range of the longitudinal velocity. It shows that the resulting model of Deep EDMD can capture the changes of the velocities in the longitudinal and lateral directions, which means the model is effective at a wide operating range. Also, the root mean square errors (RMSEs) are used to evaluate the performance, i.e.,

$$\text{RMSE} = \sqrt{\frac{1}{P} \sum_{i=1}^{P} (x_i - \hat{x}_i)},$$

(30)

where $P = 1000$ is the predicted step. The RMSEs corresponding to the velocity trajectories of all the algorithms in Fig. 4 are listed in Table III. The results show that the RMSEs of the proposed Deep EDMD for $v_x, v_y, \psi$ are the smallest ones among all the approaches in the two testing cases. That is to say, the results obtained with Deep EDMD are better than those with EDMD, MLP, LSTM, and LTV-LDM.

To fully compare our method with MLP and LSTM, the training and validation errors collected in the training process are shown in Fig. 5. The training results show that our method has smaller convergent loss value and higher convergent speed compared with MLP and LSTM.
The prediction performance of the vehicle dynamics with different methods

(a) Longitudinal velocity

(b) Lateral velocity

Fig. 4. Long-term prediction results of the vehicle dynamics with EDMD, Deep EDMD, MLP, LSTM, and LTV-LDM. The true value trajectories in (a) and (b) are sampled from the testing data sets and are chosen to get a wide range in longitudinal velocity.

Fig. 5. The training and validation reconstruction errors of Deep EDMD, LSTM, and MLP. Compared with MLP and LSTM, the proposed Deep EDMD converges faster and has minor training and validation losses.

In order to show the robustness of our approach, the obtained models of Deep EDMD and MLP with an additional random layer and bias, are used to generate the state predictions with 100 repeated times. The average value of the predicted 100 states was regarded as the final prediction and the prediction RMSEs are drawn in Table IV. The results show that the prediction errors with Deep EDMD are much smaller than those with MLP. This is due to the Koopman operator with the EDMD framework adopted in the proposed approach. Indeed, the worst scenario of Deep EDMD with the random layer can be regarded as an EDMD framework. Hence, the robustness of the results can be guaranteed.

Velocity profile tracking with DE-MPC: To further validate the potential of the proposed Deep EDMD approach, we have designed the DE-MPC controller for tracking a velocity reference in the CarSim/Simulink simulation environment. In the

| Table IV | THE RMSEs of the predicted states in the case with random hidden layers in the Deep EDMD and MLP |
|----------|----------------------------------------------------------------------------------------------------------------------------------|
| Testing case 1 | Testing case 2 |
| Deep EDMD (Training) | MLP (Training) | Deep EDMD (Validation) | LSTM (Training) | LSTM (Validation) | MLP (Training) | MLP (Validation) |
| $v_x$ | 0.4 | 3.86 | 0.35 | 15.56 |
| $v_y$ | 0.01 | 0.01 | 0.05 | 0.06 |
| $\psi$ | 0.01 | 0.01 | 0.01 | 0.04 |
simulation test, the control steering wheel angle, throttle, and the brake pressure are limited in the range of \([-450^\circ, 450^\circ]\), \([0, 0.2]\), and \([0, 9.1]\) MPa respectively. Also, the control increments are limited respectively to the range \([-2.25^\circ, 2.25^\circ]\), \([-0.004, 0.004]\), and \([-0.18, 0.18]\) MPa.

The designed DE-MPC is compared with two MPC methods for velocity profile tracking. The velocity profile reference was sampled from the testing data sets. The adopted two MPC methods include an MPC using an LTV (called LTV-MPC) vehicle dynamic model [26], [59] and a nonlinear MPC (NMPC) [27]. The state of the vehicle has been initialized as \([0, 0, 0]^	op\), and \(R = I_{m \times m}\), \(\rho = 10\) for all MPC methods. As for DE-MPC and LTV-MPC, \(Q = 1000 I_L \times L\) and \(Q = 1000 I_n \times n\), respectively. For NMPC, \(Q = \text{diag}(500, 500, 1000)\) since the solver was failed to solve the optimization problem using \(Q = 1000 I_n \times n\). The sampling interval in the simulation has been chosen as \(t_s = 10\) ms. All the MPC approaches track the reference under two different prediction and control horizons, i.e., \(N_p = N_c = 10\) and \(N_p = N_c = 20\). The hyper-parameters of LTV-MPC and NMPC were fine-tuned for fair comparisons. Note that almost all internal parameters of the analytic models for LTV-MPC and NMPC are actual values provided by CarSim, but these parameters are difficult to be estimated in practical applications.

The velocity profile tracking RMSEs of all MPC methods are calculated by (30) and are listed in Table V, where \(P = 2000\) is the overall simulation step. The results show that in both cases, satisfactory tracking performance can be achieved by DE-MPC. Besides, DE-MPC can obtain better tracking accuracy using a larger prediction and control horizons, while it is not obviously for LTV-MPC and NMPC. The velocity reference tracking results of all MPC methods with \(N_p = N_c = 20\) are shown in Fig. 6. At around 41 s, our approach shows significant performance improvement to LTV-MPC and NMPC. The reason is that the vehicle reached the limit of tire-road friction, leading to varying cornering stiffness values. Consequently, the models used in LTV-MPC and NMPC exhibit large mismatches with the real one, resulting in control performance degradation. In other words, the dynamics modeled by Deep EDMD has better adaptability in high maneuvering and driving-in-the-limit scenarios. Unlike LTV-MPC and NMPC, DE-MPC is more suitable for situations with large prediction and control horizons. An extra test with \(N_p = N_c = 50\) displays that DE-MPC only consumes an average of 7.67 ms for online computation at each time period, and the resulting RMSE values of \(v_x\), \(v_y\), and \(\dot{\psi}\) are 1.44, 0.22, and 0.06, respectively.

As depicted in Table VI, the average computational time of our approach grows slower with the prediction horizon when compared with NMPC and LTV-MPC. The reasons behind this are analyzed as follows. LTV-MPC has to update model parameters and the Hessian matrices in (26) at each time step in the prediction intervals, while in DE-MPC the model is time-invariant. As for NMPC, it needs to solve a nonlinear optimization problem, which is computationally intractable when using larger prediction and control horizons.

### TABLE V

|                  | DE-MPC | LTV-MPC | NMPC | DE-MPC | LTV-MPC | NMPC |
|------------------|--------|---------|------|--------|---------|------|
| \(v_x\) (m/s)   | 1.91   | 3.79    | 4.03 | 1.66   | 4.13    | 4.11 |
| \(v_y\) (m/s)   | 0.23   | 0.49    | 0.56 | 0.23   | 0.36    | 0.46 |
| \(\dot{\psi}\) (rad/s) | 0.07   | 0.10    | 0.10 | 0.07   | 0.11    | 0.07 |

### VII. Conclusion

In this paper, we propose a novel data-driven vehicle modeling and control approach based on deep neural networks with an interpretable Koopman operator. In the proposed approach, a deep learning-based extended dynamic mode decomposition (Deep EDMD) algorithm is presented to learn a finite basis function set of the Koopman operator. Based on the dynamic
model learned by Deep EDMD, a novel model predictive controller called DE-MPC is presented for velocity profile tracking control of autonomous vehicles. In the proposed algorithm, deep neural networks serve as the encoder and decoder in the framework of EDMD for learning the Koopman operator. Simulation studies with data sets obtained from a high-fidelity CarSim environment have been performed, including the comparative modeling methods with EDMD, MLP, LSTM, and LTV-SDM. The simulation results show the effectiveness of our approach and the advantages in terms of modeling accuracy. We also tested the performance of DE-MPC for realizing velocity profile tracking in the CarSim environment. DE-MPC demonstrates strong adaptability for real-time control in high maneuvering and driving-in-the-limit scenarios. Future research will utilize the learned model for trajectory tracking in a real-world experimental platform of autonomous vehicles.

REFERENCES

[1] B. Paden, M. Çap, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” IEEE Trans. Intell. Veh., vol. 1, no. 1, pp. 33–55, Mar. 2016.

[2] J. Chen, W. Zhan, and M. Tomizuka, “Autonomous driving motion planning with constrained iterative LQR,” IEEE Trans. Intell. Veh., vol. 4, no. 2, pp. 244–254, Jun. 2019.

[3] J. Zhang, Q. Su, B. Tang, C. Wang, and Y. Li, “DPSNet: Multitask learning using geometry reasoning for scene depth and semantics,” IEEE Trans. Neural Netw. Learn. Syst., to be published, doi: 10.1109/TNNLS.2021.3107362.

[4] X. Zhang, Y. Jiang, Y. Lu, and X. Xu, “A receding-horizon reinforcement learning approach for kinodynamic motion planning of autonomous vehicles,” IEEE Trans. Intell. Veh., to be published, doi: 10.1109/TIV.2021.3131641.

[5] M. Liu, K. Chour, S. Rathnam, and S. Darbha, “Lateral control of an autonomous and connected following vehicle with limited preview information,” IEEE Trans. Intell. Veh., vol. 6, no. 3, pp. 406–418, Sep. 2021.

[6] Z. Xu and X. Jiao, “Robust control of connected cruise vehicle platoon with uncertain human driving reaction time,” IEEE Trans. Intell. Veh., to be published, doi: 10.1109/TIV.2022.3146284.

[7] R. B. Dieter Schramm and M. Hiller, “Experimental platform of autonomous vehicles,” IEEE Trans. Veh. Technol., vol. 656, pp. 5–28, 2010.

[8] X. Li and W. Yu, “Dynamic system identification via recurrent multilayer perceptrons,” Inf. Sci., vol. 147, no. 1–4, pp. 45–63, 2002.

[9] J. Morton, F. D. Witherden, and M. J. Kochenderfer, “Deep variational Kalman filter for learning dynamics,” IFAC-PapersOnLine, vol. 54, no. 20, pp. 617–623, 2021.

[10] M. Al-Gabalawy, “Deep learning for Koopman operator optimal control,” Automatica, vol. 137, 2022, Art. no. 110114.

[11] V. Cibulka, T. Hanš, and M. Hromčík, “Data-driven identification of vehicle dynamics using Koopman operator,” in Proc. IEEE 22nd Int. Conf. Process Control Process, 2019, pp. 167–172.

[12] R. Wang, Y. Han, and U. Vaidya, “Deep Koopman data-driven optimal control framework for autonomous racing,” to be published, doi: 10.13140/RG.2.2.21512.75526.

[13] J. Xu et al., “An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2020, pp. 5049–5055.

[14] M. Korda and I. Mezić, “A hybrid lateral dynamics model combining data-driven and physical models for vehicle control applications,” IFAC-PapersOnLine, vol. 54, no. 20, pp. 106–113, 2019.

[15] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, “Predictive active steering control for autonomous vehicle systems,” IEEE Trans. Control Syst. Technol., vol. 15, no. 3, pp. 566–580, May 2007.

[16] S. Z. Yong, G. Herrmann, and J. Z. Jiang, “Approximation of dynamical systems by continuous time recurrent neural networks,” Neural Netw., vol. 6, no. 6, pp. 801–806, 1993.

[17] X. Li and W. Yu, “Dynamic system identification via recurrent multilayer perceptrons,” Inf. Sci., vol. 147, no. 1–4, pp. 45–63, 2002.

[18] G. Garimella, J. Funke, C. Wang, and M. Kobilarov, “Neural network modeling for steering control of an autonomous vehicle,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 2609–2615.

[19] B. O. Koopman, “Hamiltonian systems and transformation in hilbert space,” Proc. Nat. Acad. Sci. USA Amer, vol. 17, no. 5, pp. 315–318, 1931.

[20] X. Jiao, “Dynamic system identification via recurrent multi layer perceptrons,” Inf. Sci., vol. 147, no. 1–4, pp. 45–63, 2002.

[21] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, “On dynamic mode decomposition: Theory and applications,” J. Comput. Dyn., vol. 1, no. 2, pp. 391–421, 2014.

[22] B. O. Koopman, “Hamiltonian systems and transformation in hilbert space,” Proc. Nat. Acad. Sci. USA Amer, vol. 17, no. 5, pp. 315–318, 1931.
[42] Z. Ping, Z. Yin, X. Li, Y. Liu, and T. Yang, “Deep Koopman model predictive control for enhancing transient stability in power grids,” *Int. J. Robust Nonlinear Control*, vol. 31, no. 6, pp. 1964–1978, 2021.

[43] N. Geneva and N. Zabaras, “Transformers for modeling physical systems,” *Neural Netw.*, vol. 146, pp. 272–289, 2022.

[44] A. Avila and J. Mezić, “Data-driven analysis and forecasting of highway traffic dynamics,” *Nature Commun.*, vol. 11, no. 1, pp. 1–16, 2020.

[45] B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz, “Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition,” *J. Neurosci. Methods*, vol. 258, pp. 1–15, 2016.

[46] H. Wu, F. Nüské, F. Paul, S. Klus, P. Koltaï, and F. Noé, “Variational Koopman models: Slow collective variables and molecular kinetics from short off-equilibrium simulations,” *J. Chem. Phys.*, vol. 146, no. 15, 2017, Art. no. 154104.

[47] A. Mardt, L. Pasquali, H. Wu, and F. Noé, “VAMPnets for deep learning of molecular kinetics,” *Nature Commun.*, vol. 9, no. 1, pp. 1–11, 2018.

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

[49] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, “Data-driven control of soft robots using koopman operator theory,” *IEEE Trans. Robot.*, vol. 37, no. 3, pp. 948–961, 2021.

[50] V. Cibulka, T. Hanžl, M. Korda, and M. Hromčík, “Model predictive control of a vehicle using Koopman operator,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 4228–4233, 2020.

[51] M. Švec, Š. Ileš, and J. Matuško, “Model predictive control of vehicle dynamics based on the Koopman operator with extended dynamic mode decomposition,” in Proc. 22nd IEEE Int. Conf. Ind. Technol., 2021, vol. 1, pp. 68–73.

[52] R. N. Jazar, *Vehicle Dynamics: Theory and Application*. Berlin, Germany: Springer, 2017.

[53] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in Proc. 27th Int. Conf. Mach. Learn., 2010, pp. 807–814.

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

[55] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[57] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[58] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous distributed systems,” 2016, arXiv:1603.04467.

[59] P. Falcone, F. Borrelli, H. E. Tseng, J. Asgari, and D. Hrovat, “Linear time-varying model predictive control and its application to active steering systems: Stability analysis and experimental validation,” *Int. J. Robust Nonlinear Control: IFAC-Affiliated J.*, vol. 18, no. 8, pp. 862–875, 2008.

Xinglong Zhang (Member, IEEE) received the Ph.D. degree in system and control from the Politecnico di Milano, Milan, Italy, 2018. He is currently an Associate Professor with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, China. His research interests include safe reinforcement learning, model predictive control, and their applications in automotive systems.

Xin Xu (Senior Member, IEEE) received the Ph.D. degree in control science and engineering from the College of Mechatronics and Automation, National University of Defense Technology (NUDT), Changsha, China, in 2002. He is currently a Visiting Professor with The Hong Kong Polytechnic University, Hong Kong, University of Alberta, Edmonton, AB, Canada, University of Guelph, Guelph, ON, Canada, and University of Strathclyde, Glasgow, U.K. He is currently a Professor with the College of Intelligence Science and Technology, NUDT. His research interests include intelligent control, reinforcement learning, approximate dynamic programming, machine learning, robotics, and autonomous vehicles. Dr. Xu was the recipient of the National Science Fund for Outstanding Youth in China and the Second-Class National Natural Science Award of China. He was an Associate Editor or the Guest Editor of Information Sciences, *International Journal of Robotics and Automation, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, Intelligent Automation and Soft Computing, International Journal of Adaptive Control and Signal Processing, and Acta Automatica Sinica*. He is also a Member of the IEEE CIS Technical Committee on Approximate Dynamic Programming and Reinforcement Learning and the IEEE RAS Technical Committee on Robot Learning.

Xueqing Liu received the master’s degree from the School of Automotive Engineering, Chang’an University, Xi’an, China, in 2015. He is currently a Software Engineer with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, China. His research interests include model identification and control of dynamic systems, and the design, analysis, and applications of dynamic systems.

Xiaohong Li received the master’s degree from the College of Information Science and Engineering, Northeastern University, Shenyang, China, 2015, and the master’s degree in 2018 from the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, China, where he is currently working towards the Ph.D. degree with the College of Intelligence Science and Technology.

Yongqian Xiao received the master’s degree in mechanical engineering from the College of Mechanical Electronical and Engineering, Central South University, Changsha, China, in 2018. He is currently working toward the Ph.D. degree with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, China. His research interests include machine learning, reinforcement learning, and autonomous vehicles.