2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

DDK: A Deep Koopman Approach for Longitudinal and Lateral Control of Autonomous Ground Vehicles

Yongqian Xiao[1], Xinglong Zhang[1], Xin Xu[1], Yang Lu[1], Junxiang Li[1]

Abstract— **Autonomous driving has attracted lots of attention in recent years. For some tasks, e.g., trajectory prediction, motion planning, and trajectory tracking, an accurate vehicle model can reduce the difficulty of these tasks and improve task completion performance. Prior works focused on parameter estimation of physical models or modeling nonlinear dynamics using neural networks. Still, these methods rely on internal parameters of vehicles or are not friendly for control due to the strong nonlinearity of models. This paper proposes a data-driven method to approximate vehicle dynamics based on the Koopman operator. The resulting model is an interpretable linear time-invariant model, facilitating controller design and solving related optimization problems. In the proposed approach, the state transition matrix is constructed based on the learned Koopman eigenvalues, while the input matrix is trained as a tensor. Based on the resulting model, a linear model predictive controller is designed to implement coupled longitudinal and lateral trajectory tracking. Simulations and experiments, including vehicle dynamics modeling and coupled longitudinal and lateral trajectory tracking, are performed in a high-fidelity CarSim environment and a real vehicle platform. An oil-driven D-Class SUV is selected in the simulation, while a real electric SUV is utilized in the experiment. Simulation and experiment results illustrate that the model of the nonlinear vehicle dynamics can be identified effectively via the proposed method, and high-quality trajectory tracking performance can be obtained with the resulting model.**

I. INTRODUCTION

Vehicle dynamics modeling and control are two important parts of autonomous driving. For vehicle dynamics, classic kinematics models are usually adopted for low-speed scenes [1]. On the contrary, dynamics models are suitable for high-speed scenarios and even high-maneuvering situations, such as drifting [2]. Time-varying and nonlinear vehicle dynamics are too complicated to obtain precise approximation models, so that some decent analytical models are established, e.g., 14-DOF dynamic model [3], separated longitudinal or lateral dynamics [4], Pacejka tire model [5], and linear time-varying dynamics [6].

Vicente et al. [7] adopted a least-squares method to approximate vehicle dynamics in a linear data-driven manner, but the resulting model showed a large prediction error. Recurrent neural networks were adopted to estimate the side-slip angle for a simplified analytic kinematics [8]. Spielberg et al. [9] constructed a vehicle dynamics with a multi-layer perception (MLP), and a complex nonlinear controller was designed to validate the identified model. Da et al. [10] studied the influences of different neural network structures on the identification performances of the longitudinal vehicle dynamics. Besides, some works estimated vehicle internal parameters to obtain the physical model, such as estimating vehicle sideslip, tire force, and cornering stiffness [11]–[13]. These estimation approaches rely on physical vehicle models [14].

The Koopman operator has received lots of attention as a tool for identifying and analyzing nonlinear systems. Generally, it is a linear operator in infinite dimensional Hilbert space unless the state space is a finite set [15]. Therefore, many approaches have been proposed and developed to approximate the Koopman operator in an acceptable finite dimension in which dynamic mode decomposition (DMD) [16] and extended DMD (EDMD) [17]–[20]. EDMD and kernel-based DMD [21] usually perform better than DMD since they lift states to an appropriate higher-dimension space by kernel functions. Appropriate kernel functions critically decide the approximating effectiveness, but they are arduous to be designed, especially for complicated nonlinear systems. As a result, neural networks are naturally introduced into taking the place of kernel functions [22] [23] [24]. In [25], the nonlinearity in the control input was encoded by an MLP.

There are several works based on the Koopman operator for modeling, controlling, or planning of autonomous vehicles [26]–[29]. Cibulka et al. [26] adopt EDMD to model a bicycle model. Based on the model, a linear model predictive control (MPC) was designed to track velocity profiles [30]. However, the simulation environment was not realistic enough and showed large tracking errors. Similar to [26] and [30], a bicycle model was modeled with EDMD and controlled by an MPC [31]. The aforementioned three works are based on the modeling and control methods in [32]. The above methods are validated based on models described by analytic equations, but real vehicle dynamics are much more complex. Xiao et al. [27] modeled a vehicle dynamics from a high-fidelity simulator CarSim using an autoencoder architecture based on EDMD. Meanwhile, a linear MPC was designed for longitudinal and lateral velocities and yaw rate profile tracking.

The approaches mentioned above have been well explored in vehicle dynamics modeling and velocity profile tracking control, but two aspects are still worth studying. One is to improve the modeling precision of the latent dynamics. The other is that the previous models only include the longitu-

The work was supported in part by the National Natural Science Foundation of China under Grant 62003361, U21A20518, and 62103431.(Corresponding authors: Xinglong Zhang, Xin Xu.)

[1] Yongqian Xiao, Xinglong Zhang, Xin Xu, Yang Lu, Junxiang Li are with the College of Intelligence Science and Technology, National University of Defense Technology, Changsha, 410073, P. R. China (e-mail: zhangxinglong18@nudt.edu.cn, xinxu@nudt.edu.cn).

dinal and lateral velocities and yaw rate as the state [27], [33], which make the trajectory tracking task a nonlinear optimization problem since the mapping from the latent state (velocities) to vehicle position is nonlinear.

Motivated by the above two requirements and inspired by Deep EDMD [27], and Deep Koopman [23], this paper proposes a deep direct Koopman (DDK) approach for approximating the Koopman operator of vehicle dynamics. The *direct* reflects in several aspects. One is that DDK directly learns the Koopman eigenvalues. The identified vehicle dynamics is described in linear time-invariant (LTI) form with a block-diagonal state transition matrix constructed by the eigenvalues. In addition, DDK directly handles the original state as the former several elements of the Koopman eigenfunctions so that an interpretable subsystem can be extracted from the identified dynamics to improve the interpretability. Finally, the vehicle pose in the vehicle coordinate is directly added to be a part of the vehicle state. Consequently, the coupled longitudinal and lateral trajectory tracking task can be formed as a liner optimization problem and be met using linear optimal controllers. Note that we cannot concatenate the original state with the features outputted by the encoder for systems with unknown states, such as systems [34], [35] with pixel-level measurements.

The main contributions of this work are as follows.

- A deep learning approach called DDK is proposed to learn the Koopman eigenvalues and input matrix for approximating the Koopman operator of vehicle dynamics. The resulting vehicle dynamics is an interpretable LTI system with good prediction capability.
- The resulting model includes vehicle poses and velocities information. A linear MPC is implemented for coupled longitudinal and lateral trajectory tracking based on the resulting model. Simulations and experiments are implemented in a high-fidelity CarSim platform and a real electronic SUV to verify the proposed approach.

II. DDK FOR MODELING VEHICLE DYNAMICS

In this work, a four-wheel vehicle dynamics model is utilized to carry out the simulation and analysis [27], which is given as follows:

$$s_{t+1} = f\left(s_t, u_t\right), \tag{1}$$

where $s \in \mathbb{R}^n$ denotes the vehicle state vector consisting of poses (x, y, ψ), and the associated velocities $(v_x, v_y, \dot{\psi})$. Poses include the longitudinal and lateral position, and yaw angle in the vehicle coordinate system, and the associated velocities including the longitudinal and lateral velocities, and yaw rate. $u_t \in \mathbb{R}^m$ is the control inputs including the steering and the engine. $f(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the nonlinear vehicle state transition function. In this paper, steering and engine denote the steering wheel angle and accelerator for the real electric vehicle. As for the oil-driven vehicle in CarSim, the steering represents the front-wheel angle, and the engine consists of the throttle opening and brake pressure.

Construction of DDK: For discrete-time vehicle dynamics of (1), the Koopman operator can be described as follows:

$$\varphi(s_{k+1}) = (\mathcal{K}\varphi)(s_k, u_k), \tag{2}$$

where $\mathcal{K}$ is the Koopman operator, which is usually an infinite-dimensional linear operator in Hilbert space $\mathcal{H}$, φ is the observable functions. Therefore, it is essential to approximate the Koopman operator in an appropriate finite-dimensional state space. In DDK, the resulting K-order approximated vehicle dynamics is described as

$$\varphi(s_{k+1}) = A\varphi(s_k) + Bu_k, \tag{3}$$

where $A \in \mathbb{R}^{K\times K}$ and $B \in \mathbb{R}^{K\times m}$ are system matrices, $\varphi(s_k)$ refers to the state of the identified dynamics at time k. Let $T = [A\ B]$, $\bar{\varphi}(s_k) = [\varphi^\top(s_k)\ u_k^\top]^\top$, then T is the former K rows of the approximation of Koopman operator. We only focus on the former K rows of $\bar{\varphi}(s_{k+1})$ since the prediction of the future control u_{k+1} is not used in controller design [32].

As described above, our purpose is to approximate the Koopman operator, which includes the observable function φ and the system matrices T of the latent dynamics. The neural network framework of DDK is depicted in Fig. 1. DDK adopts an auto-encoder structure, where the encoder and the decoder play the roles of the observable function φ and the Koopman modes. The Koopman eigenvalues are learned and used to parameterize block matrices for establishing the block-diagonal state transition matrix A.

To approximate the Koopman operator in a finite dimension, the Koopman eigenfunctions are approximated with features consisting of the original state followed by characteristics extracted by the encoder,

$$\varphi(s_k) = [s_k^\top; \phi_e^\top(s_k, \theta_e)]^\top \tag{4}$$

where ϕ_e denotes the encoder parameterized with θ_e.

System evolution in the latent state space actually is scaling in the directions of the eigenfunctions in which the scaling magnitude equals the eigenvalues [36]. DDK parameterizes the state transition matrix by learning the Koopman eigenvalues directly. For situations only with real eigenvalues, $\Lambda = \{R_1, R_2, \cdots, R_K\}$, A is constructed as a diagonal matrix $A = diag\{R_1, R_2, \cdots, R_K\}$, where R_i denotes the i-th eigenvalue. Generally, nonlinear systems have conjugate complex pairs. The eigenvalues are described as $\Lambda = \{a_1, b_1, \cdots, a_g, b_g, R_1, \cdots, R_h\}$, where $C_j = a_j \pm b_j i$ denotes the j-th conjugate complex pair, g is the number of conjugate complex pairs of eigenvalues, and h represents the number of real eigenvalues. Consequently, A is constructed into a block-diagonal matrix as follows:

$$A = \text{b-diag}\{\mathfrak{B}_1, \cdots, \mathfrak{B}_g, R_1, \cdots, R_h\}, \tag{5}$$

where $\mathfrak{B}_j \in \mathbb{R}^{2\times 2}$ is the matrix corresponding to the j-th conjugate complex pair, and it is built as

$$\mathfrak{B}_j = \begin{bmatrix} a_j & b_j \\ -b_j & a_j \end{bmatrix}, \tag{6}$$

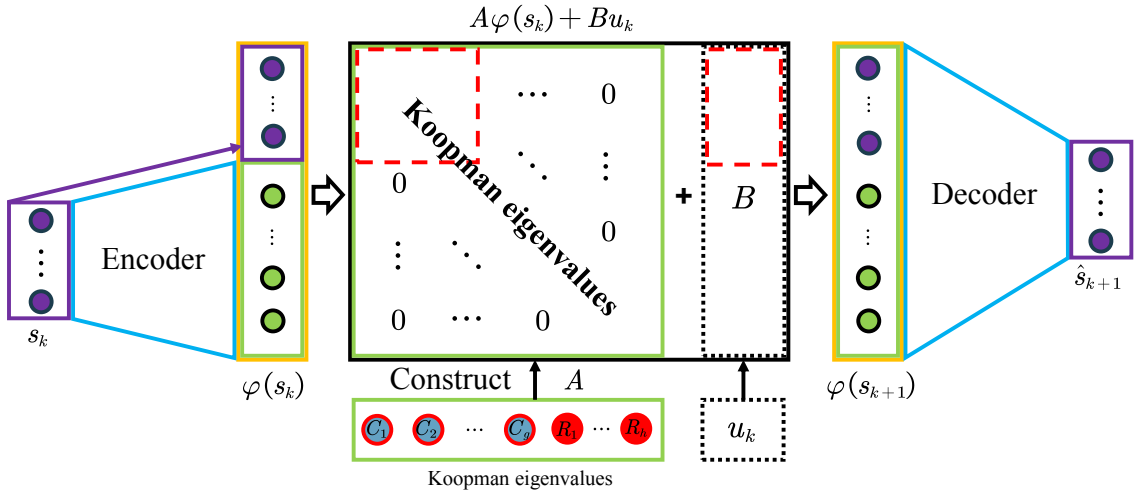


Fig. 1. The neural network framework of DDK. This framework adopts the structure of auto-encoder (AE), and the original state is concatenated to the outputted features by the encoder. $\varphi(s_k)$ is an approximation of the Koopman eigenfunctions, and it is also the latent state of the approximated linear vehicle dynamics. Especially, the state transition matrix is constructed with the learned Koopman eigenvalues via structuring a block-diagonal matrix. Red dashed wireframes express the interpretable subsystem described by the n-th leading principal submatrix of A and B. This framework is also suitable for unforced dynamics while u_k equals $\mathbf{0}_{n\times 1}$ constantly. R_k denotes the k-th real eigenvalue while C_j denotes the j-th conjugate complex pair, and c indicates that adjacent c frames of vehicle states are concatenated as a new state.

where a_j and b_j denote the real and imaginary parts of C_j. A is the K-order approximation of the Koopman operator and $K = 2g + h$. Determining K, g, and h remains an open issue. In DDK, g and h are allocated according to K as follows:

$$\begin{aligned} h &= K \mod 2, \\ g &= \frac{K-h}{2}. \end{aligned} \tag{7}$$

In DDK, the system matrices A and B are learned as constant matrices after training. That is, the resulting dynamics (3) is a linear time-invariant system. It can be easily extended to p steps linear evolution:

$$\hat{\varphi}(s_{k+p}) = A^p \varphi(s_k) + \sum_{i=1}^{p} A^{i-1} B u_{k+p-i}. \tag{8}$$

The Koopman modes implement full-state observable in eigenfunctions basis, and the performance can be improved by representing them with neural networks [24],

$$\hat{s}_k = \phi_d(\varphi(s_k)), \tag{9}$$

where ϕ_d represents the decoder parameterized by θ_d.

Since A is block-diagonal, the former n elements of $\varphi(x_k)$ match the physical vehicle state. As a result, when n is an even number, a subsystem can be extracted from (3) shown in Fig. 1 with red dashed frames. This subsystem is represented as $s_{k+1} = A_n s_k + B_n u_k$, where A_n and B_n are the n-th leading principal submatrix of A and B, respectively. When n is not an even number, we can adjust the order of real eigenvalues and complex conjugate pairs of eigenvalues to extract the subsystem.

Loss functions: In this work, the designed loss functions refer to prior works [23], [24], [27]. These loss functions include reconstruction loss, linear loss, multi-step reconstruction loss, and regularization loss. They constrain the reconstruction, linearity, multi-step prediction performances, and avoid over-fitting, respectively,

$$\begin{aligned} \mathcal{L}_r &= \frac{1}{p}\sum_{i=1}^{p} \|s_{k+i} - \hat{s}_{k+i}\|_2^2), \\ \mathcal{L}_l &= \frac{1}{p}\sum_{i=1}^{p} \|\varphi(s_{k+i}) - \hat{\varphi}(s_{k+i})\|_2^2, \\ \mathcal{L}_{mr} &= \sum_{i=1}^{p} \|s_{k+i} - \phi_d(\hat{\varphi}(s_{k+i}))\|_2^2, \\ l_2 &= \|\theta_e\|_2^2 + \|\theta_d\|_2^2 + \|\Lambda\|_2^2 + \|B\|_2^2. \end{aligned} \tag{10}$$

The approximation process of the Koopman operator is built as an optimization problem to minimize the following weighted loss,

$$\mathcal{L} = \alpha_1 \mathcal{L}_r + \alpha_2 \mathcal{L}_l + \alpha_3 \mathcal{L}_{mr} + \alpha_4 l_2. \tag{11}$$

where α_i is the corresponding weight to different losses that expresses the importance of each loss. Implementation steps are outlined in algorithm 1.

MPC based on DDK vehicle model (DDK-MPC): Based on the learned DDK vehicle dynamics (3), a linear MPC is designed for coupled longitudinal and lateral trajectory tracking of autonomous vehicles. That is, it demands the autonomous vehicle to follow the reference trajectory, including poses and associated velocities along the time sequence. The design of DDK-MPC is similar to [27], and the main difference is that we need to transform the reference trajectory to the vehicle coordinate at each time step. To account for the limits of space, please refer to [27] for more details of the linear MPC design.

III. Simulation Validation

In this section, the CarSim environment provides training, validation, and testing datasets. DDK is validated in two

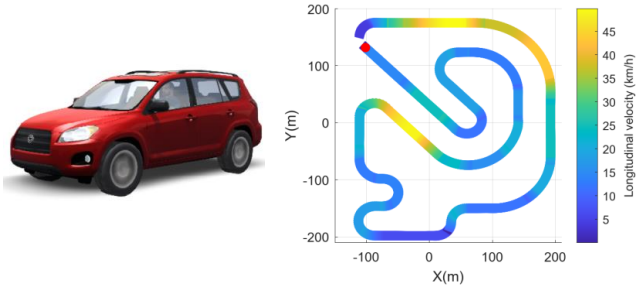
Algorithm 1 Implementation steps for DDK

Initialize θ_e, θ_d, Λ, B, time step p, α_i, $i = 1, \cdots, 4$, batch size b_s, learning rate β.
repeat
  Sample a batch of sequences and transform them to randomly selected vehicle coordinate systems.
  Obtain the Koopman eigenfunctions $\varphi(s_{0:p})$ with (4) and reconstruction states $\hat{s}_{0:p}$ with (9).
  Construct A based on the eigenvalues Λ with (5).
  Perform p steps Koopman operator with (8) and reconstruct them with (9) to get $\hat{\varphi}(s_{1:p})$ and $\phi_d(\hat{\varphi}(s_{1:p}))$.
  Calculate the weigted loss with (11).
  Update θ_e, θ_d, Λ, B with an Adam optimizer.
until The epoch terminated

aspects, i.e. multi-step prediction and coupled longitudinal and lateral trajectory tracking combining CarSim.

A. Datasets collection and preprocessing

The datasets consist of 38 episodes and are collected by combining CarSim 2019 with MATLAB/Simulink using sampling time $t_s = 50ms$. The length of each episode is about 9100 time steps. As shown in Fig. 2, an oil-driven D-Class SUV was adopted to collect the datasets. Each episode is a trajectory with different velocities in which trajectories were generated by a proportional controller for longitudinal velocity control and a pure pursuit (PP) controller for lateral steering control. Values of the front wheel angle are in the range of $\zeta \in [-25°, 25°]$, and values of the engine η consists of the brake pressure $[0, 1]MPa$ and throttle opening $[0, 1]$. Note that positive values of the engine denote the throttle opening while negative values indicate the brake pressure and vice versa.



(a) (b)

Fig. 2. The vehicle for data collecting and reference trajectory. (a) The oil-driven D-Class SUV in CarSim. (b) An example reference trajectory.

The preprocessing contains two stages. The first stage merges the throttle opening and brake into a single variable. It normalizes the longitudinal and lateral velocities, and the yaw rate to the range of $[-2, 2]$, while the control variables are normalized into the range of $[-1, 1]$. The second stage transforms trajectories to the local coordinate so that the trajectory tracking task is still a linear optimization problem based on (3) and normalizes them to $[-2, 2]$.

TABLE I
HYPERPARAMETERS IN DDK.

H-param	Value	H-param	Value	H-param	Value
Learning rate	10^{-4}	Batch size	512	c	1
K	16	p	60	α_1	1.0
α_2	1.0	α_3	1.0	α_4	10^{-6}

Main hyperparameters are outlined in Table I. The dimension of latent states K is an important hyperparameter that too small values lead to poor identification performance, but too big values cost a large calculation resulting in the trouble of real-time requirements in control on the contrary.

The encoder and decoder are implemented with two MLPs that have the structure of $[n \cdot c, 64, 128, 64, K]$ and $[K, 128, 64, 64, n]$, respectively. This work was trained using the Python API in PyTorch-Lightning framework with an Adam optimizer based on an NVIDIA GeForce GTX 2080 Ti GPU. The corresponding MPC algorithm is implemented in MATLAB/Simulink 2020a combining CarSim 2019 environment with an Intel i7-11700KF@3.6GHz.

B. Model prediction

In this subsection, the identifying capability of DDK and the trajectory tracking performance DDK-MPC are verified separately. MLP and LSTM used in [37], Deep EDMD [27], EDMD [26], [32], and LS approaches are adopted to identify the same vehicle dynamics and to compare with the proposed DDK. We did not choose Deep Koopman [23] as a comparison since we did not achieve good performance for modeling the vehicle dynamics using Deep Koopman. LTV-MPC [6], [38] and PP [39] are selected for comparisons in the aspect of trajectory tracking control.

Deep EDMD: Deep EDMD was implemented with the same hyperparameters with DDK, such as the structure of the autoencoder, p, and K.

EDMD: As the same in [26], we adopted thin-plate spline radial basis functions to span the observable space. The kernel centers κ were selected randomly in the uniform distribution $U(-2, 2)$.

LS: The purpose of adopting LS as a comparison is to show the effectiveness of EDMD, since EDMD is also a least-square method.

LSTM: LSTM takes $[\hat{x}_i^\top \ u_i^\top]^\top$ as the input to predict the next state, where $\hat{x}_i$ is the prediction state based on $\hat{x}_{i-1}$, and we have $\hat{x}_0 = x_0$. The structure of LSTM consists of three vanilla LSTM cells and an MLP. The structure of the MLP in LSTM was designed as $[n \ 128 \ 128 \ n]$. LSTM also adopts multi-step loss function as the proposed DDK,

$$\mathcal{L}_{\text{lstm}} = \frac{\gamma_1}{p}\sum_{i=1}^{p}\|x_i - \hat{x}_i\|_2^2 + \frac{\gamma_2}{p}\sum_{i=1}^{p}\|x_i - \hat{x}_i\|_\infty + \gamma_3\|\theta_L\|_2^2, \tag{12}$$

where γ_i for $i = 1, 2, 3$ are weighting scalars. θ_L denotes all the weights and biases of LSTM. DDK, LSTM, and MLP

TABLE II

RMSEs OF 60 PREDICTION STEPS (3S) FOR DIFFERENT METHODS

Method	x (m)	y (m)	ψ (rad)	v_x (m/s)	v_y (m/s)	$\dot{\psi}$ (rad/s)
DDK	**0.385**	**0.159**	**0.013**	**0.167**	**0.007**	**0.005**
DeepEDMD	1.067	0.631	0.035	0.377	0.012	0.009
EDMD	1.532	0.833	0.043	0.424	0.044	0.032
LS	7.838	0.573	0.049	0.506	0.048	0.037
MLP	7.216	0.512	0.059	1.146	0.051	0.030
LSTM	0.832	0.297	0.031	0.560	0.016	0.013

adopt the same learning rate and the step p in loss functions. **MLP:** In MLP, we also take $[\hat{x}_i^\top \ u_i^\top]^\top$ as the input and the structure of MLP was designed as $[n+m\ 32\ 64\ 128\ 128\ 64\ 32\ n]$. Besides, MLP adopt the same p-step loss functions as LSTM in (12).

As outlined in Table II, root-mean-square error (RMSE) of 60 prediction steps is adopted to evaluate the prediction performance of different methods. Each method predicts and calculates its RMSE using the whole testing dataset containing about 5000 sequences. The results show that DDK obtains the best prediction performance.

C. Coupled longitudinal and lateral trajectory tracking

To further validate the proposed DDK method, coupled longitudinal and lateral trajectory tracking simulation was implemented based on Simulink/CarSim environment. Besides, an MPC based on LTV vehicle dynamics was implemented as a comparison. Pure-pursuit was also implemented in this section. MPCs based on EDMD and Deep EDMD did not select for comparisons since their lateral errors at curves on the road are too large to complete the whole trajectory tracking task. Diagonal penalty matrices of states and controls for DDK-MPC and LTV-MPC was chosen as $Q = \text{diag}\{q_1, ..., q_n\}$, $\bar{R} = \text{diag}\{r_1, r_2\}$ where $q_i = 20$ for $i = \{1, 5, 6\}$, $q_j = 1000$ for $j = \{2, 3, 4\}$, $r_1 = 5$, and $r_2 = 10000$. For LTV-MPC, it has the same parameter tuning way as DDK-MPC and the best performance is chosen. PP only tracks the lateral direction with the preview distance $\delta_d = k_d v_x + \delta_{df}$, where $k_d = 0.4$ and $\delta_{df} = 4m$. For the PP controller, there is a proportional feedback rule for tracking longitudinal velocity. Other hyperparameters are shown in Table III. It is worth noting that the engine is piecewise to represent the brake pressure and throttle opening according to the sign then anti-normalize them to the original range for applying to the CarSim car.

The testing reference trajectory is shown in Fig. 2 (c). The maximum and mean absolute errors of coupled longitudinal and lateral trajectory tracking results with different controllers are given in Table IV, where $\mathcal{E}_d$ denotes the lateral tracking error. The results show that our method can obtain the best tracking performance, and the average lateral error is only 6.5cm. The simulation results of all trajectories in the training, testing, and validation datasets can be gained and

TABLE III

MPC HYPERPARAMETERS IN THE SIMULATION.

H-param	Value	H-param	Value
ε_{min}	0	ε_{max}	100
u_{min}	$[-1.0, -1.0]^\top$	u_{max}	$[1.0, 1.0]^\top$
Δu_{min}	$[-0.5, -0.5]^\top$	Δu_{max}	$[0.5, 0.5]^\top$
ρ	10	t_s	$50ms$
N_p	30	N_c	30

TABLE IV

MAXIMUM AND MEAN ABSOLUTE ERRORS FOR COUPLED LONGITUDINAL AND LATERAL TRACKING IN CARSIM SIMULATION

	Method	$\mathcal{E}_d$ (m)	X (m)	Y (m)	ψ (rad)	v_x (m/s)	v_y (m/s)	$\dot{\psi}$ (rad/s)
Mean	**DDK-MPC**	**0.065**	**0.033**	**0.046**	**0.003**	**0.247**	**0.006**	**0.005**
	LTV-MPC	1.134	0.596	0.769	0.034	0.443	0.008	0.007
	PP	0.380	0.227	0.252	0.012	0.308	0.010	0.008
Max	**DDK-MPC**	**0.375**	**0.283**	**0.276**	**0.027**	**1.032**	**0.049**	**0.040**
	LTV-MPC	7.836	5.087	7.797	0.439	9.653	0.213	0.174
	PP	1.706	1.496	1.643	0.123	1.951	0.171	0.148

recovered using our control code[1].

The average one-step solving time of DDK-MPC, LTV-MPC, and PP are 7.31ms, 13.85ms, and 0.36ms, respectively. PP had the shortest solving time since PP is a forward controller and does not deal with constraints. Compared with LTV-MPC, DDK-MPC spent less solving time since the DDK vehicle dynamics is linear time-invariant.

IV. EXPERIMENT

In this section, a real vehicle is utilized to collect datasets for validating the capabilities of modeling and tracking control. As shown in Fig. 3, road scenes include straight roads, U-turns, roundabouts, left and right turns. In experiments, except the hyper-parameters given in Table V, other hyper-parameters are the same as the simulation. Besides, DDK-MPC was implemented using C++ with an Intel i7-11800H@2.3GHz.

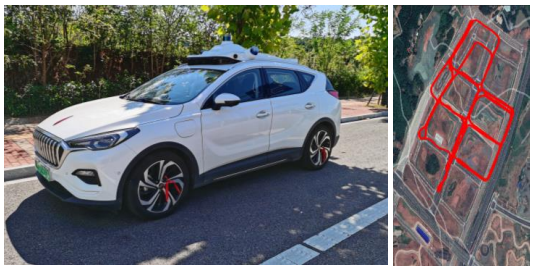


Fig. 3. HongQi EHS3 SUV and one example trajectory. Differences with CarSim data include sampling interval is $20ms$ and the engine is represented by acceleration instead of the throttle opening and brake pressure.

As outlined in Table VI, RMSEs of prediction with different prediction horizons are calculated by the whole testing

[1]The CarSim datasets, CarSim config file, control code, DDK model, and all tracking results can be obtained: https://github.com/yongqianxiao/DDK

TABLE V

HYPERPARAMETERS IN THE EXPERIMENT.

H-param	Value	H-param	Value
K	12	p	80
Δu_{min}	$[-0.05,-0.05]^\top$	Δu_{max}	$[0.05,0.05]^\top$
N_p	50	N_c	20
R	diag{10000,10000}	t_s	20ms

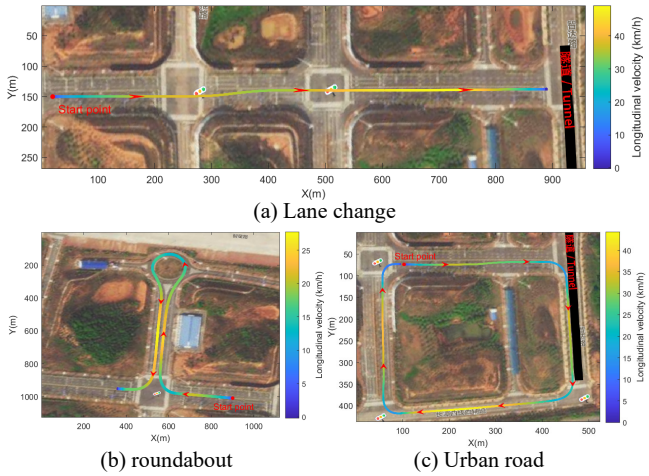
TABLE VI

RMSES OF PREDICTION WITH DIFFERENT HORIZONS

Steps $(20ms)$	x (m)	y (m)	ψ (rad)	v_x (m/s)	v_y (m/s)	$\dot{\psi}$ (rad/s)
80(1.6s)	0.299	0.169	0.027	0.027	0.023	0.030
180(3.6s)	0.467	0.472	0.064	0.055	0.033	0.041
240(4.8s)	0.726	0.666	0.080	0.073	0.034	0.041

dataset. Prediction results demonstrate that DDK receives acceptable RMSE even for prediction more than 3s.

We focus more on coupled longitudinal and lateral trajectory tracking control in the experiment. As shown in Fig. 4, we adopt five urban scenes in the experiment, and the largest longitudinal velocity is about 50km/h. Maximum and mean absolute errors are given in Table VII, where ΔT denotes the time error between the reference time and the spent tracking control time. We can know that the vehicle controlled by DDK-MPC was basically synchronized with the reference trajectories in the longitudinal direction. The results show a small mean lateral tracking error, though the maximum errors are slightly larger when the vehicle turns. Besides, the average one-step solving time is only about **0.98ms**. Fig. 5 shows that DDK-MPC is tracking the reference trajectory collected by the human driver. Full videos of more experiment scenes are publicly available [2].



(a) Lane change

(b) roundabout (c) Urban road

Fig. 4. Five experiment scenes. Satellite imagery did not update, leading to differences in the experiment video.

[2]https://yongqianxiao.github.io/2022/09/12/Experiment-videos-of-DDK/

TABLE VII

MAXIMUM AND MEAN ABSOLUTE ERRORS FOR COUPLED LONGITUDINAL AND LATERAL TRACKING IN HONGQI EHS3

	Scene Fig. 4	$\mathcal{E}_d$ (m)	X (m)	Y (m)	ψ (rad)	v_x (m/s)	v_y (m/s)	$\dot{\psi}$ (rad/s)	ΔT (s)
Mean	(a)	0.144	0.112	0.077	0.005	0.043	0.063	0.011	0.04
	(b)	0.175	0.128	0.092	0.011	0.054	0.054	0.011	0.52
	(c)	0.103	0.057	0.071	0.005	0.051	0.030	0.006	0.56
Max	(a)	0.341	0.290	0.230	0.035	0.240	0.215	0.043	-
	(b)	0.745	0.740	0.380	0.052	0.227	0.239	0.048	-
	(c)	0.792	0.350	0.710	0.035	0.308	0.144	0.025	-

Fig. 5. Experiment video screenshot. The left picture shows the reference trajectory collected by the human driver, while the right picture shows the synchronicity of DDK-MPC for tracking the reference trajectory.

V. CONCLUSION

A pure data-driven deep learning approach called DDK is proposed to approximate the Koopman operator of vehicle dynamics for coupled longitudinal and lateral trajectory tracking. DDK directly learns the time-invariant Koopman eigenvalues to construct a block-diagonal state transition matrix. In addition, the vehicle state matches the former n elements of the latent state. An interpretable subsystem can be extracted from the identified latent dynamics and utilized to be the prediction model for control. An MPC is designed to implement coupled longitudinal and lateral trajectory tracking in a high-fidelity vehicle dynamics environment. Simulation results validate the feasibility of DDK-MPC for trajectory tracking under the condition of real-time requirements. Finally, DDK is utilized to model the dynamics of a real electric SUV, and corresponding coupled longitudinal and lateral trajectory tracking experiments are implemented. Future research will focus on high-efficiency motion planning based on the identified vehicle dynamics.

REFERENCES

[1] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: Theory and applications," *arXiv preprint arXiv:1312.0041*, 2013.

[2] J. Y. Goh, T. Goel, and J. Christian Gerdes, "Toward automated vehicle control beyond the stability limits: Drifting along a general path," *Journal of Dynamic Systems, Measurement, and Control*, vol. 142, no. 2, 2020.

[3] R. Zheng, C. Liu, and Q. Guo, "A decision-making method for autonomous vehicles based on simulation and reinforcement learning," in *2013 International Conference on Machine Learning and Cybernetics*, vol. 1. IEEE, 2013, pp. 362–369.

[4] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[5] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious nmpc with gaussian process dynamics for autonomous miniature race cars," in *2018 European Control Conference (ECC)*. IEEE, 2018, pp. 1341–1348.

[6] 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," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 18, no. 8, pp. 862–875, 2008.

[7] B. A. H. Vicente, S. S. James, and S. R. Anderson, "Linear system identification versus physical modeling of lateral-longitudinal vehicle dynamics," *IEEE Transactions on Control Systems Technology*, 2020.

[8] T. Gräber, S. Lupberger, M. Unterreiner, and D. Schramm, "A hybrid approach to side-slip angle estimation with recurrent neural networks and kinematic vehicle models," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 1, pp. 39–47, 2018.

[9] N. A. Spielberg, M. Brown, N. R. Kapania, J. C. Kegelman, and J. C. Gerdes, "Neural network vehicle models for high-performance automated driving," *Science Robotics*, vol. 4, no. 28, p. eaaw1975, 2019.

[10] M. Da Lio, D. Bortoluzzi, and G. P. Rosati Papini, "Modelling longitudinal vehicle dynamics with neural networks," *Vehicle System Dynamics*, vol. 58, no. 11, pp. 1675–1693, 2020.

[11] B.-C. Chen and F.-C. Hsieh, "Sideslip angle estimation using extended kalman filter," *Vehicle System Dynamics*, vol. 46, no. S1, pp. 353–364, 2008.

[12] J. Bechtoff and R. Isermann, "Cornering stiffness and sideslip angle estimation for integrated vehicle dynamics control," *IFAC-PapersOnLine*, vol. 49, no. 11, pp. 297–304, 2016.

[13] B. L. Boada, M. J. L. Boada, and V. Diaz, "Vehicle sideslip angle measurement based on sensor data fusion using an integrated anfis and an unscented kalman filter algorithm," *Mechanical Systems and Signal Processing*, vol. 72, pp. 832–845, 2016.

[14] H. Guo, D. Cao, H. Chen, C. Lv, H. Wang, and S. Yang, "Vehicle dynamic state estimation: state of the art schemes and perspectives," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 418–431, 2018.

[15] A. Mauroy, I. Mezić, and Y. Susuki, *The Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*. Springer Nature, 2020, vol. 484.

[16] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of fluid mechanics*, vol. 656, pp. 5–28, 2010.

[17] 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.

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

[19] M. O. Williams, M. S. Hemati, S. T. Dawson, I. G. Kevrekidis, and C. W. Rowley, "Extending data-driven koopman analysis to actuated systems," *IFAC-PapersOnLine*, vol. 49, no. 18, pp. 704–709, 2016.

[20] G. Mamakoukas, M. L. Castano, X. Tan, and T. D. Murphey, "Derivative-based koopman operators for real-time control of robotic systems," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2173–2192, 2021.

[21] I. G. Kevrekidis, C. W. Rowley, and M. O. Williams, "A kernel-based method for data-driven koopman spectral analysis," *Journal of Computational Dynamics*, vol. 2, no. 2, pp. 247–265, 2016.

[22] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 10, p. 103111, 2017.

[23] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature communications*, vol. 9, no. 1, pp. 1–10, 2018.

[24] S. E. Otto and C. W. Rowley, "Linearly recurrent autoencoder networks for learning dynamics," *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 1, pp. 558–593, 2019.

[25] H. Shi and M. Q.-H. Meng, "Deep koopman operator with control for nonlinear systems," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7700–7707, 2022.

[26] V. Cibulka, T. Haniš, and M. Hromčík, "Data-driven identification of vehicle dynamics using koopman operator," in *2019 22nd International Conference on Process Control (PC19)*. IEEE, 2019, pp. 167–172.

[27] Y. Xiao, X. Zhang, X. Xu, X. Liu, and J. Liu, "Deep neural networks with koopman operators for modeling and control of autonomous vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 135–146, 2023.

[28] G. Gutow and J. D. Rogers, "Koopman operator method for chance-constrained motion primitive planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1572–1578, 2020.

[29] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[30] V. Cibulka, T. Haniš, M. Korda, and M. Hromčík, "Model predictive control of a vehicle using koopman operator," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 4228–4233, 2020.

[31] M. Švec, Š. Ileš, and J. Matuško, "Model predictive control of vehicle dynamics based on the koopman operator with extended dynamic mode decomposition," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1. IEEE, 2021, pp. 68–73.

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

[33] G. Rödönyi, R. Tóth, D. Pup, Á. Kisari, Z. Vígh, P. Kőrös, and J. Bokor, "Data-driven linear parameter-varying modelling of the steering dynamics of an autonomous car," *IFAC-PapersOnLine*, vol. 54, no. 8, pp. 20–26, 2021.

[34] Y. Xiao, X. Xu, and Q. Lin, "Cknet: A convolutional neural network based on koopman operator for modeling latent dynamics from pixels," *arXiv preprint arXiv:2102.10205*, 2021.

[35] B. van der Heijden, L. Ferranti, J. Kober, and R. Babuška, "DeepKoCo: Efficient latent planning with an invariant Koopman representation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. [Online]. Available: https://arxiv.org/abs/2011.12690

[36] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of koopman eigenfunctions for control," *Machine Learning: Science and Technology*, vol. 2, no. 3, p. 035023, 2021.

[37] J. Xu, Q. Luo, K. Xu, X. Xiao, S. Yu, J. Hu, J. Miao, and J. Wang, "An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 5049–5056.

[38] F. Lin, Y. Zhang, Y. Zhao, G. Yin, H. Zhang, and K. Wang, "Trajectory tracking of autonomous vehicle with the fusion of dyc and longitudinal–lateral control," *Chinese Journal of Mechanical Engineering*, vol. 32, no. 1, pp. 1–16, 2019.

[39] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.