Driver Behavior Modeling Using Game Engine and Real Vehicle: A Learning-Based Approach

Ziran Wang, *Member, IEEE*, Xishun Liao, *Student Member, IEEE*, Chao Wang, *Student Member, IEEE*, David Oswald, Guoyuan Wu, *Senior Member, IEEE*, Kanok Boriboonsomsin, *Member, IEEE*, Matthew J. Barth, *Fellow, IEEE*, Kyungtae Han, *Senior Member, IEEE*, BaekGyu Kim, *Member, IEEE*, and Prashant Tiwari

***Abstract*—As a good example of Advanced Driver-Assistance Systems (ADAS), Advisory Speed Assistance (ASA) helps improve driving safety and possibly energy efficiency by showing advisory speed to the driver of an intelligent vehicle. However, driver-based speed tracking errors often emerge, due to the perception and reaction delay, as well as imperfect vehicle control, degrading the effectiveness of ASA system. In this study, we propose a learning-based approach to modeling driver behavior, aiming to predict and compensate for the speed tracking errors in real time. Subject drivers are first classified into different types according to their driving behaviors using the k-nearest neighbors (*k*-NN) algorithm. A nonlinear autoregressive (NAR) neural network is then adopted to predict the speed tracking errors generated by each driver. A specific traffic scenario has been created in a Unity game engine-based driving simulator platform, where ASA system provides advisory driving speed to the driver via a head-up display (HUD). A human-in-the-loop simulation study is conducted by 17 volunteer drivers, revealing a 53% reduction in the speed error variance and a 3% reduction in the energy consumption with the compensation of the speed tracking errors. The results are further validated by a field implementation with a real passenger vehicle.**

***Index Terms*—Driver behavior, game engine, machine learning, neural network, Advanced Driver-Assistance Systems (ADAS), human-in-the-loop simulation, field implementation.**

I. Introduction and Background

A. Introduction

RAPID growth has been witnessed worldwide in the development of intelligent vehicles due to the improvements in perception, communication, and computation technologies. As an enabler of intelligent vehicles, Advanced Driver-Assistance Systems (ADAS) aim to support drivers by providing either warnings to reduce risk exposure or advisory information to relieve drivers' burden on some of the driving tasks [1]. Many studies on ADAS have been conducted over the past several decades, such as pedestrian detection [2], vehicle overtaking [3], forward collision avoidance, and adaptive cruise control [4].

As a typical example of ADAS, Advisory Speed Assistance (ASA) systems are developed that recommend speed limits to bound the vehicle speed on highways, or generate advisory speed profiles for human drivers to track on signalized arterials [5]–[7]. The advisory information is delivered to the driver via a human-machine interface (HMI), allowing him/her to control the longitudinal speed to improve safety, mobility, and/or environment factors.

However, human tracking errors are inevitably introduced into ASA systems, since it is impossible for drivers to track advisory speed profiles perfectly. It is concluded by one of our previous studies that the speed tracking errors may account for as high as 12% degradation in system performance (i.e., energy consumption) of the original ASA design [8]. Therefore, in this study, we focus on modeling driver behavior and compensate for the speed tracking errors in a personalized manner.

B. Driver Behavior Modeling

Driver behavior plays a significant role in the development of ADAS in terms of driving safety as well as vehicle energy management. Numerous research and development efforts have been focused on the topic of driver behavior identification and classification. Martinez *et al*. reviewed a variety of algorithms with an emphasis on machine learning approaches, and also discussed applications of driver behavior recognition [9].

Many researchers proposed to use rule-based approaches for driver behavior modeling, which are usually defined for particular events or factors, such as acceleration/deceleration, turning movements, lane changes, road types, vehicle gaps, and energy consumption [10]–[12]. One major drawback of the rule-based approaches is the complexity of encoding driver behavior by a fixed number of rules, since it may vary significantly with drivers and scenarios, and too many parameters need to be considered. Also, the quality of a rule-based algorithm is highly affected by the selection of those parameters, which heavily relies on the algorithm designer's expertise [9].

Manuscript received August 19, 2019; revised October 31, 2019, January 19, 2020, and March 25, 2020; accepted April 8, 2020. Date of publication May 6, 2020; date of current version November 23, 2020. This work was supported by the "Digital Twin" project of Toyota Motor North America, InfoTech Labs. (*Corresponding author: Ziran Wang*.)

Ziran Wang, Kyungtae Han, BaekGyu Kim, and Prashant Tiwari are with InfoTech Labs, Toyota Motor North America, Mountain View, CA 94043 USA (e-mail: ryanwang11@hotmail.com; kyungtae.han@toyota.com; baekgyu.kim@toyota.com; prashant.tiwari@toyota.com).

Xishun Liao, Chao Wang, David Oswald, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J. Barth are with the Center for Environmental Research and Technology, University of California Riverside, Riverside, CA 92507 USA (e-mail: xliao016@ucr.edu; cwang061@ucr.edu; roswa001@ucr.edu; gywu@cert.ucr.edu; kanok@cert.ucr.edu; barth@ece.ucr.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIV.2020.2991948

More recently, there has been an increasing amount of research that utilizes learning-based approaches for driver behavior modeling, given their advantages to process massive data. Specifically, both unsupervised and supervised machine learning algorithms have been proposed. Since unsupervised algorithms do not require the underlying procedures to be understood, modeling of driver behavior is achieved through statistical analysis of the input signals inherent to the algorithms. Constantinescu et al. developed two alternative algorithms to model driver behavior, based on hierarchical cluster analysis (HCA) and principal component analysis (PCA), respectively [13]. The Gaussian Mixture Model (GMM) was implemented by Miyajima et al. to analyze the car-following behavior [14], and was also adopted by Butakov and Ioannou to develop personalized driver/vehicle lane change models [15]. The Bayesian learning was implemented by McCall and Trivedi to evaluate critical situations regarding braking assistance [16], while a Bayesian regression model was developed by Mudgal et al. to characterize driver behavior at roundabouts [17].

Supervised algorithms imply knowledge of the data used for training driver behavior models. The neural network and Markov model have been adopted as two major supervised learning approaches for driver behavior modeling. Xu et al. developed a distal learning control framework with two feedforward neural networks [18], while Augustynowicz applied an Elman-type neural network to identify driver behavior based on the vehicle speed and throttle pedal position [19]. In addition, Wei et al. proposed an end-to-end lane change behavior prediction model with the deep residual neural network [20]. For Markov models, since future states only depend on current states and the transition can be categorized by transition probability matrices, their benefits in terms of modeling time-continuous driver behavior were defended by Guardiola et al. [21]. The Dynamical Markov Model [22], and Hidden Markov Model [23]–[25] were developed by different researchers to model driver behavior, with the consideration of drivers' actions being best captured as a sequence of control steps.

C. Game Engine

Game engines are software systems that allow software developers to design video games, and typically consist of a rendering engine (i.e., "renderer") for 2-D or 3-D graphics, a physics engine for collision detection and response, and a scene graph for the management of multiple elements (e.g., models, sound, scripting, threading, etc.). Along with the development of a game engine, its function has been broadened to a wider scope: data visualization, training, medical, and military use.

More recently, the game engine becomes a popular option for researchers worldwide to prototype connected vehicles [26], ADAS [27], and autonomous vehicles [28]. LGSVL is developed by the Unity game engine to facilitate testing and development of autonomous driving software systems, and it provides real-time outputs from sensors like camera, LiDAR, RADAR, GPS, and IMU [29], [30]. CARLA is implemented as an open-source layer over UnrealEngine4 (UE4), from the ground up to support training, prototyping and validation of autonomous driving models [31], [32].

Although there are several other options to conduct driver behavior modeling and simulation, such as the commercial automotive simulation software PreScan and CarSim [33], [34], the Unity game engine is adopted in this study given the following strengths [26]:

- *Graphics rendering and visualization:* Since the game engine is originally for developing 3D video games, it has a strong capability of graphics rendering and visualization. It streamlines the demonstration of the proposed intelligent vehicle technology to the general public.
- *Integration with driving simulators:* The game engine provides easy access to switch the input equipment, which makes it possible to integrate with driving simulator hardware, as well as virtual reality (VR) and augmented reality (AR) equipment for future development.
- *Asset store:* The Unity game engine has an official asset store where developers and users can upload and download different Unity assets, allowing them to develop a game environment based on existing work instead of building everything from scratch.
- *Documentation and community:* Unity provides thorough, well-organized, and easy-to-read documentation covering how to use each component in Unity. An online community has been set up for all Unity users to ask and answer questions.

D. Paper Structure

The remainder of this paper is organized as follows: In the problem statement of Section II, the major contributions of this study compared to existing work are presented. Section III explains the detailed design of the proposed ASA in the game engine. Section IV develops a learning-based approach to modeling driver behavior of our ASA, aiming to predict and compensate for the speed tracking errors by using historical data generated by volunteer drivers. Results of the human-in-the-loop simulation using the Unity game engine are given in Section V, where further validation is performed with a real passenger vehicle described in Section VI. Finally, the paper is concluded with some future directions in Section VII.

II. Problem Statement

In this study, we develop an ASA system using a head-up display (HUD) design with the Unity game engine and a HMI design on a real passenger vehicle, allowing various drivers to either conduct the human-in-the-loop simulation or drive a real vehicle to track the advisory speed. However, given that the driver cannot perfectly follow the advisory speed command generated by ASA system, speed tracking errors will always be generated. Therefore, we further propose a learning-based approach to modeling driver behavior, aiming to predict and compensate for the speed tracking errors generated by each driver in a personalized way.

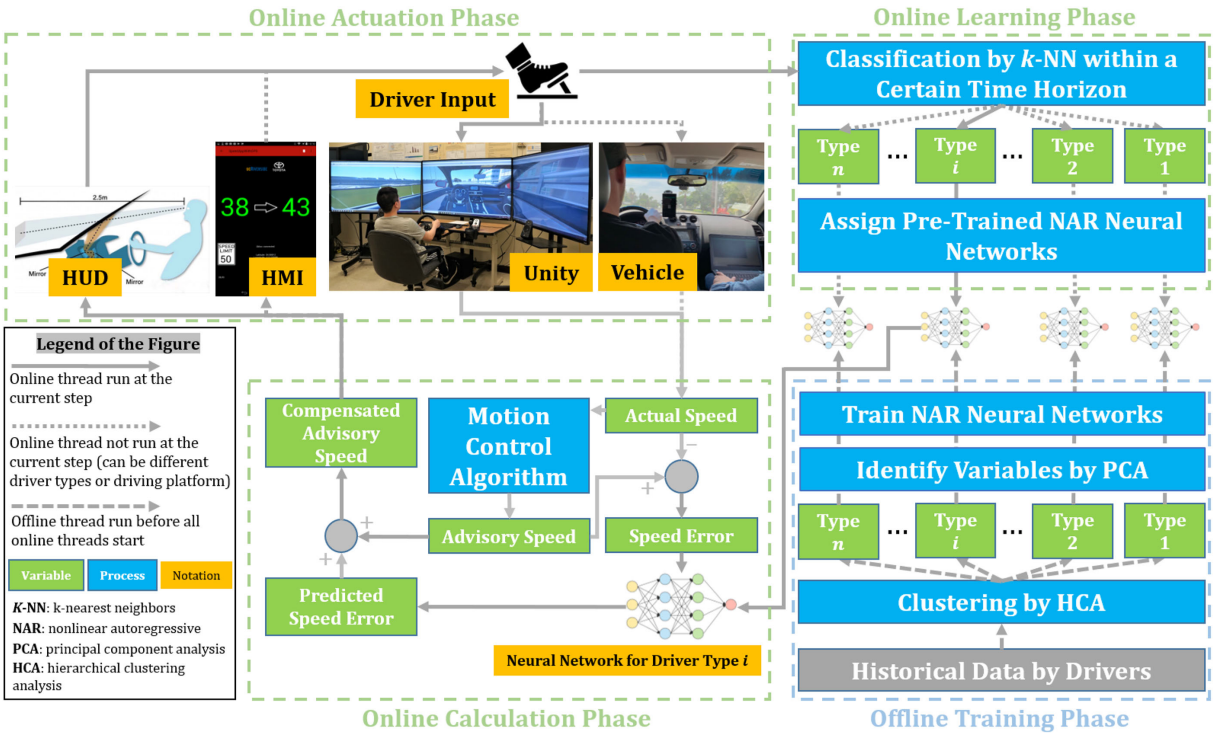


Fig. 1. System architecture of the proposed learning-based driver behavior modeling system.

Compared to many previous studies on driver behavior modeling (including our previous work [8]), the major contributions of this study are listed below:

- *Personalized advisory:* By adopting the learning-based approach with a nonlinear autoregressive (NAR) neural network, the proposed system can classify different drivers into certain types and provide personalized advisory information. To this end, the accuracy and effectiveness of the proposed ASA system can be greatly improved, compared to existing systems that only provide general advisory information to all drivers.
- *Integrated vehicle system:* Instead of only focusing on predicting driver behavior, we design an integrated vehicle system which includes a motion controller, a learning model, and user interfaces (both HUD and HMI). This integrated vehicle system is also presented by two proofs of concept.
- *Multi-platform online validation:* Different from some existing work that validate their driver behavior models using synthetical data with numerical simulation, we design an ASA system with HUD in the game engine and invite various volunteer drivers to conduct the human-in-the-loop simulation on the driving simulator platform, so the improvement of the proposed driver behavior model can be observed in real time. Furthermore, we validate the effectiveness of our driver behavior model using a real passenger vehicle. This study allows various drivers to participate and test the proposed model in two different platforms, so the results will be more convincing than any previously proposed studies which simply conduct computer simulation.
- *Wide applicability:* Although only a specific cooperative merging scenario is considered in this study, the proposed learning-based driver behavior model can be applied to various traffic scenarios, as long as the advisory speed is helpful for human drivers. For example, the similar concept can be applied to scenarios like speed harmonization on highways [35], eco-driving on signalized corridors [6], or stop-free unsignalized intersection management [36].

The general architecture of the proposed ASA system is shown in Fig. 1. The whole system can be broken down into four different phases, including the online actuation phase, the online calculation phase, the online learning phase, and the offline training phase. The former two phases are introduced in Section III, while the latter two phases are depicted in Section IV.

III. ASA in the Game Engine

This section first introduces the cooperative merging algorithm (i.e., the online calculation phase) which generates the advisory speed for the driver. Then the development of the online actuation phase within the game engine environment is described, which is later applied to the real passenger vehicle as well. Lastly, driver's inputs to the Unity game engine based on the HUD design are introduced as a simplified vehicle dynamics model. Note that Unity is not a software specifically designed for vehicle simulation, so the low-level vehicle dynamics model needs to be customized by users to reach the best simulation performance.

Fig. 2. Cooperative merging scenario at on-ramp built in Unity.

Fig. 3. Design of HUD-based ASA in Unity.

A. ASA for Cooperative Merging

In this study, we focus on an on-ramp cooperative merging scenario, where the controlled intelligent vehicles can collaborate with each other (by adjusting longitudinal speeds only) to perform on-ramp merging safely and smoothly through V2X communication. Our previously proposed feedforward/feedback motion control algorithm is applied here to calculate the advisory longitudinal speed [37]. Basically, once the controlled intelligent vehicle i is assigned with a leader vehicle j to follow by the sequencing protocol developed in [38], it retrieves information of this leader vehicle through V2X communication, which includes length l_j, longitudinal speed v_j, and longitudinal position r_j. Then, the proposed motion controller takes those inputs as well as the vehicle dynamics data from the ego vehicle, and computes a reference acceleration a_{ref} for the ego vehicle at the next time step by the following algorithm

$$a_{ref}(t+\delta t) = -\alpha_{ij}k_{ij}\cdot\left[\left(r_i(t) - r_j\left(t-\tau_{ij}(t)\right) + l_j\right.\right.$$
$$\left.\left.+v_i(t)\cdot\left(t_{ij}^g(t)+\tau_{ij}(t)\right)\right) + \gamma_i\cdot\left(v_i(t) - v_j\left(t-\tau_{ij}(t)\right)\right)\right] \tag{1}$$

where δt is the length of each time step, α_{ij} denotes the value of adjacency matrix, k_{ij} and γ_i are control gains, $\tau_{ij}(t)$ denotes the time-variant communication delay between two vehicles, and $t_{ij}^g(t)$ is the time-variant desired time gap between two vehicles. The advisory speed displayed on ASA system can be computed as

$$v_i\left(t+\delta t\right) = v_i(t) + a_{ref}\left(t+\delta t\right)\cdot\delta t \tag{2}$$

where $v_i(t+\delta t)$ is the advisory speed shown to the driver, and $v_i(t)$ is the current speed of the ego vehicle.

B. HUD Design in the Game Engine

A real-world on-ramp cooperative merging scenario is built in the Unity game engine as shown in Fig. 2. This particular 267-meter on-ramp is modeled based on a realistic on-ramp in Mountain View, California, which connects E Middlefield Rd and California State Route 237 westbound. The red on-ramp vehicle tries to merge into the vehicle string on the main line, between those two dark blue vehicles. As can be seen from Fig. 2, the on-ramp and the main line have different elevations (thus potentially obstructing the line of sight), so V2X-based ASA can provide the driver with helpful information regarding vehicles traveling on the other lane.

In this study, the red on-ramp vehicle is controlled by the human driver on the driving simulator platform. As shown in Fig. 3, the driver of this vehicle is provided a HUD system, which is projected on the windshield right above the dashboard. HUD originates from a pilot being able to view information with the head positioned "up" and looking forward, instead of looking down at lower instruments. Although it was initially developed for military aviation, HUD is now widely used in commercial aircraft and automobiles. On production vehicles, HUD is usually enabled to offer speedometer, tachometer and navigation system displays [39].

The HUD design in this study is developed by the "Canvas" in Unity, which is a game object with a canvas component on it. All HUD elements, such as the speed information and vehicle indicator, are children of this canvas. The canvas is shown as a rectangle in the scene view of Unity, allowing us to easily position our HUD design on the windshield and right above the dashboard. Since we attach the game object canvas as a child of the game object ramp vehicle, the HUD design becomes the grandchild of the ramp vehicle and is fixed on that position of the windshield.

Additionally, the preceding vehicle (with respect to the ego vehicle) is identified through V2X communication and the on-board camera. It is marked with a "TARGET LEADER" indicator on top of its roof (which is also projected as HUD on the windshield), notifying the driver of which vehicle to follow during the merging process. Note that we also design the side mirrors and the rear-view mirror in the vehicle cabin, allowing the driver to observe vehicles running behind while conducting the merging behavior.

C. Driver Inputs Based on ASA

Once the advisory speed $v_i(t+\delta t)$ is computed by Eq. (2) and displayed to the driver by the aforementioned HUD design, the driver of the controlled on-ramp vehicle needs to input executions to Unity to track that advisory speed in the longitudinal direction, while keeping the vehicle at the center

of the current lane. Therefore, we develop a vehicle controller to transfer driver lateral and longitudinal inputs into vehicle dynamics based on the game object "WheelCollider" in Unity. Basically, a "collider" in Unity defines the shape of an object for physical collisions. "WheelCollider" is a special "collider" designed for vehicles with wheels in Unity, which has built-in collision detection, wheel physics, and a slip-based tire friction model. Note that in the game engine we only model a simplified version of the vehicle powertrain, which can map the driver's lateral input to the steering angle of vehicle's front wheels, and map the driver's longitudinal input to the rotational speed of vehicle's two or four wheels.

1) Driver Lateral Input: The steering input of the vehicle is denoted as $u_x \in [-1, 1]$, which allows two front wheels to steer along the local x-axis in Unity. The steering angle θ can be calculated by

$$\theta = \theta_{\max} \cdot u_x \tag{3}$$

where $\theta_{\max}$ is the user-defined maximum steering angle of the front wheels. The steering m angle θ can then be passed to the front wheels to steer the vehicle by the "WheelColliders $[i]$.steerAngle $= \theta$" command in Unity's C# scripting API, where $i = 0, 1$ denotes the front left and right wheels.

2) Driver Longitudinal Input: The throttle or brake inputs of the vehicle is denoted as $u_y \in [-1, 1]$, which allows the vehicle to move along the local y-axis by thrust/brake torque, generated by its two wheels (if the vehicle is either front-wheel drive or rear-wheel drive) or all four wheels (if the vehicle is four-wheel drive). When $u_y \in (0, 1]$, it is considered as the driver stepping on the accelerator pedal. Alternatively, the driver is assumed to step on the brake pedal when $u_y \in [-1, 0)$.

The thrust τ_{thrust} and brake torque τ_{brake} can be calculated by

$$\tau_{thrust} = \frac{\tau_{tmax}}{n_{wheels}} \cdot u_y, \text{if} u_y \in [0, 1] \tag{4}$$

$$\tau_{brake} = \tau_{bmax} \cdot u_y, \text{if} u_y \in [-1, 0) \tag{5}$$

where τ_{tmax} denotes the full thrust torque of the vehicle over all wheels, τ_{bmax} denotes the full brake torque of each wheel, $n_{wheels} = 2$ if the vehicle is either front-wheel drive or rear-wheel drive, and $n_{wheels} = 4$ if the vehicle is four-wheel drive. These two equations are built on the assumptions that the thrust torque can be distributed to all wheels evenly, and there is no kinetic energy loss in the whole process (for both thrust and brake).

The thrust torque τ_{thrust} can then be passed to the wheels to accelerate the vehicle by the "WheelColliders $[i]$.motorTorque $= \tau_{thrust}$" command in the scripting API, where $i = 0, 1, 2, 3$ denotes the front left, front right, rear left and rear right wheels, respectively. Similarly, the brake torque τ_{brake} can be passed to each wheel to decelerate the vehicle by the "WheelColliders $[i]$.brakeTorque $= \tau_{brake}$" command in the scripting API.

Fig. 4. Driving simulator platform built on Logitech Racing Wheel and Unity.

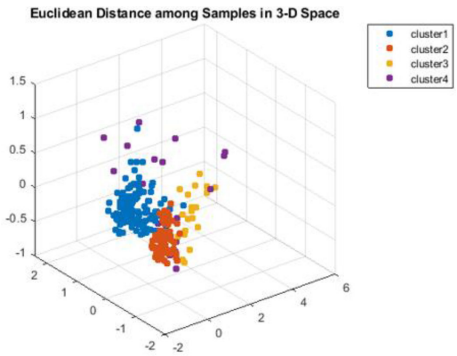


Fig. 5. HCA cluster visualization in 3-D space.

IV. Learning-Based Driver Behavior Modeling

In this section, we introduce a learning-based approach to modeling driver behavior and hence predicting the speed tracking errors. In the offline training phase, by learning the behavior of different drivers based on historical data, we cluster drivers into N types and train N neural networks. In the online learning phase, a driver will be classified into one of the preset types and assigned to the associated neural network according to his/her driving style within a certain time horizon. In the closed loop, once the neural network takes the speed tracking errors as inputs, it predicts the driving error in the next time step. Then the advisory speed calculated by the merging algorithm can be compensated by the error prediction module, thus shown on HUD.

A. Driver Data Generation

The human-in-the-loop simulation is conducted to allow volunteer drivers to test the HUD design in Unity, so training data for the driver type clustering can be generated. We also compare these results with the ones generated when ASA is disabled. Since Unity provides easy access for users to switch their gaming inputs among keyboard, mouse and joystick, it is we can integrate the plug-and-play feature in the test platform. As shown in Fig. 4, a driving simulator platform is built based on a Windows desktop (processor Intel Core i7-7700K @ 4.20 GHz, memory 64.0 GB), Unity (version 2018.3.12f), and a Logitech G27 Racing Wheel (model W-U0001).

We invite 17 volunteers with real-world driving experience to participate in this human-in-the-loop simulation. To reduce any system biases in the simulation results, volunteers are chosen from various backgrounds: 1 senior driver (age >50), 2 mid-age drivers ($30<$ age $<=50$), and 14 young drivers (age $<=30$); 15 male drivers and 2 female drivers. The drivers are guided to try their best to follow ASA during the simulation, so that the ego vehicle can perform the cooperative merging maneuvers in a smoother way compared to the scenario when no ASA is provided.

At the very beginning, each volunteer drives the vehicle on the simulator multiple times to collect data for training. Note that two different merging scenarios are developed in Unity, where the driver is randomly asked to drive either the ramp vehicle or the mainline vehicle. Additionally, only one volunteer at a time is allowed to enter the room of the simulator. Therefore, the volunteer will not have any prior knowledge regarding the traffic scenario, so his/her driving behavior totally depends on how well he/she can track the HUD-based ASA.

B. Driver Type Clustering and Classification

This subsection aims to cluster the test subjects into different types according to the similarity of their driver behavior. For the observation of speed, four variables are measured during each run.

- The variance of speed (σ_v) describes the stability of the driving.
- The mean error of speed (μ_{Δ_v}) is the average difference between the advisory speed and actual speed, which evaluates the execution ability of the driver. Also, it distinguishes the driver who is always slower than the advisory speed from the driver who always exceeds the advisory speed.
- The absolute mean error of speed $|(\mu_{\Delta_v})|$ avoids misclassifying the driver who has a small mean error of speed, but actually drives pretty aggressively.
- The variance of the speed error (σ_{Δ_v}) is the variance of the difference between the advisory speed and the actual speed, implying the stability of the driver's execution.

Similarly, five variables in the observation of acceleration are also measured, including:

- The variance of acceleration (σ_a).
- The mean error of acceleration (μ_{Δ_a}).
- The absolute mean error of acceleration $(|\mu_{\Delta_a}|)$.
- The variance of the acceleration error (σ_{Δ_a}).
- The mean of acceleration (μ_a).

Since the number of driver type is not strictly defined in this study, an unsupervised learning approach is used to cluster the driver. The pseudocode of this HCA is stated as Algorithm 1. The Euclidean distance and Ward linkage method, which are both HCA methods, are adopted to create a hierarchical cluster tree for clustering.

We combine each driver's data as a matrix X, $X=\{X_1,\ldots,X_i,\ldots,X_n\}$, where $X_i=\{\sigma_v,\mu_{\Delta_v},|\mu_{\Delta_v}|,\mu_{\Delta_a},\mu_a,|\mu_{\Delta_a}|,\sigma_a,\sigma_{\Delta_a}\}$ and compute the Euclidean distance matrix D as

$$D=\begin{bmatrix} 0 & D_{12} & \cdots & D_{1n-1} & D_{1n} \\ D_{21} & 0 & \ldots & D_{2n-1} & D_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ D_{n-11} & D_{n-12} & \ldots & 0 & D_{n-1n} \\ D_{n1} & D_{n2} & \cdots & D_{nn-1} & 0 \end{bmatrix} \quad (6)$$

where $D_{ij}=\|X_i-X_j\|_2^2$.

Algorithm 1 HCA: Cluster the Driving Type.

Input: Matrix (X) that contains 9 variables of driver behavior and n samples.

Output: K clusters.

1: Compute the distance matrix.
2: **While** *the number of clusters > 1*
3: Merge two clusters with the smallest D_{ij};
4: Update the distance matrix;
5: Save D_{ij} and cluster ID in the stack;
6: The number of clusters is cut in half;
7: **end**
8: Separate from one cluster into several clusters based on the median distance among recorded D_{ij};

As explained in Algorithm 1, when the dendrogram is generated based on the similarity of each data sample, we cut the dendrogram by the median Euclidean distances among the samples to obtain the final clusters. After filtering out the outlier samples, all valid samples in the driver dataset are clustered into four major types. We adopt the Multidimensional Scaling (MDS) to display the driver type clustering result in 3-D space, which is shown in Fig. 5. Note that if more data samples are obtained, we might cluster them into more than these four types to achieve better performance.

Once the clustering is finished, we need to explore useful features from the data to obtain a more precise model. For instance, to compensate for the human tracking errors, we identify the most contributing variables as the object for the neural network to predict. Moreover, in high dimensions, there is little difference between the nearest and the farthest neighbor for the k-nearest neighbors (k-NN) classification using Euclidean distance because of "the curse of dimensionality", so we reduce the input variables for classification. As stated in Algorithm 2, PCA is used to transform these nine correlated variables into a set of linearly uncorrelated variables, which are called principal components. Note PCA is not utilized before the driver clustering since the computational burden is not bottlenecked by the clustering, and all the original variables are potentially helpful for the clustering.

As stated in the $n\times 9$ matrix of driver's data, we have nine types of features and n data samples. Specifically, we propose Algorithm 2 to identify the important variables to predict the speed tracking errors. According to the analytical results of PCA, the first component solely contributes 74.76%, and the second one solely contributes 18.85%. Several criteria for deciding how many components should be chosen are given in [13]: (a) the "elbow" in visual interpretation plot, (b) meaningful percentage

Algorithm 2 PCA:: Identify the Important Variables.

Input: Matrix X that contains nine variables of driver behavior.

Output: 1) Accumulate percentage of singular value (POS). 2) Correlation coefficients matrix.

1: Normalize the data matrix;
2: For each column X_j in X, $X_i = X_i - \mu$;
3: Calculate the covariance matrix
$K_{xx} = COV[X, X] = E[(X - \mu_x)(X - \mu_x)^T]$;
4: Calculate the singular values Σ and singular vector V, based on $K_{xx} = V\Sigma^2 V^T$;
5: Arrange Σ in descending order POS $= \frac{\Sigma_i}{sum(\Sigma)}$;
6: Calculate the correlation matrix (factor loading) R;

Algorithm 3 *k*-NN: Classify the New Driver.

Input: 1) The speed trajectory of the driver during the time horizon $t_{classify}$. 2) Sample data of clustered drivers. 3) The number of neighbors to be considered.

Output: The type of the user.

1: Compute the data matrix X which contains the nine variables, where $X_i = \{\sigma_v, \sigma_{\Delta v}, \sigma_a, \sigma_{\Delta a}\}$;
2: Compute the similarity between the driver and all other previous drivers, where $S_i = \|X - X_i\|_2^2$;
3: Rank S and pick out the top k samples;
4: Do majority-voting
For k samples
If (sample = type1): {type1.VOTE +1}
Else: {type2.VOTE +1}
End
If (type1.VOTE >= type2.VOTE): {Type = Type 1}
Else: Type = Type 2

of variance (80–90%), and (c) interpretable components. To meet these three criteria, we keep the first two components.

According to the correlation results, the variance of the speed errors (σ_{Δ_v}) has a good correlation with the first principal component, where $\sigma_{\Delta v}$ ranks the highest in the correlation, and the variance of speed (σ_v) stands out among the others. Having a higher correlation with the first two principal components, speed-related variables contribute much more than the acceleration on the driving behavior, so predicting and compensating for the speed errors can have a significant improvement in the execution of the advisory speed. We also notice the variance of the acceleration errors (σ_{Δ_a}) and the variance of acceleration (σ_a) play important roles, which can be considered as another two variables in the classification.

Once we obtain these different clusters by HCA and find out the important features by PCA, we can classify drivers into those clusters based on their driving behaviors during the time horizon of $t_{classify}$. By proposing the *k*-NN algorithm (as stated in Algorithm 3), we classify the driver into the same type as those that share the similar driving behavior.

C. Training the Nonlinear Autoregressive (NAR) Neural Network

Once all historical data generated by various drivers are clustered, neural network is trained to predict driver behavior.

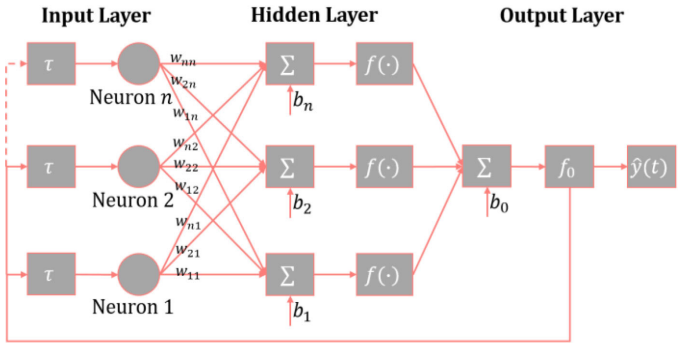


Fig. 6. Structure of NAR neural network.

The speed errors generated from the driver when tracking ASA can be considered as a time series with high variations. Since it is generally difficult to model a time series using a linear model, we adopt the NAR neural network [40] in this study. It has been proved by Lapedes and Farber [41] that, time series can always be modeled by the following NAR model

$$\hat{y}(t + \delta t) = f\{y(t), y(t - \delta t), y(t - 2\delta t), \ldots, y(t - (\tau - 1) \times \delta t)\} \tag{7}$$

where δt denotes a time step, and the speed tracking error y at time $t + \delta t$ is predicted using τ past values of the series. The structure of the NAR network can be seen as Fig. 6. To approximate the unknown function $f(\cdot)$, the neural network is trained by means of the optimization of network weights w's and neuron biases b's. The numbers of hidden layers and neurons per layer are completely flexible, which can be optimized through a trial-and-error process. Note that more neurons may complicate the system, but less neurons may restrict the generalization capability and computing power of the network.

In this study, to train the series of speed error data, we set the number of hidden layers as 2, and the number of hidden neurons as 10. The number of delays τ is set to 2, which means a total of 3 values of the speed tracking errors are used to predict the value at the next time step. The Levenberg-Marquardt back-propagation procedure (LMBP) is implemented as the learning rule of this NAR network [42]. LMBP is considered one of the fastest backpropagation-type algorithms, since it was designed to approximate the second-order derivative without computing the Hessian Matrix. The training process is conducted on the Windows desktop with processor Intel Core i7-7700K @ 4.20 GHz and 64.0 GB memory.

We evaluate our training result using the Mean Squared Error (MSE), which is the average squared difference between predictions and targets, and the Regression (R) value, which is a measurement of the correlation between output predictions and targets. For MSE, a lower value stands for a better result where zero means no error. A higher R value means a stronger correlation between the prediction and target, while zero stands for a random relationship. We pick two biggest clusters out of those four shown in Fig. 5, and split 70% for training, 15% for validation, and another 15% for testing. As shown in the Table I, two neural networks are trained with high performance, since

TABLE I
TRAINING RESULT OF NAR NEURAL NETWORK

Data Set	Catalog	Target Values	MSE	R
Driver Type 1	Training	20512	0.0147	0.9906
	Validation	4396	0.0067	0.9959
	Testing	4396	0.0073	0.9956
Driver Type 2	Training	8330	0.0166	0.9929
	Validation	1785	0.0140	0.9939
	Testing	1785	0.0191	0.9953

their MSE values are lower than 0.02 and R values are higher than 0.99.

Once the neural networks are completely trained, they can be implemented in an online manner as shown in Fig. 1. At every time step, the trained neural network (configured as a MATLAB script) takes multiple inputs through the UDP socket from either the game engine or the vehicle, computes the predicted speed error at the next time step, and sends it back through the UDP socket. Once the predicted speed error is received, it will then be compensated for the original advisory speed algorithm Eq. (2) by

$$\hat{v}_i\left(t+\delta t\right) = v_i(t) + a_{ref}\left(t+\delta t\right)\cdot\delta t + \hat{y}\left(t+\delta t\right) \qquad (8)$$

where $\hat{y}(t+\delta t)$ is the predicted error term compensated for the advisory speed. This compensated advisory speed $\hat{v}_i(t+\delta t)$ is the value that is eventually displayed to the driver.

V. RESULTS OF HUMAN-IN-THE-LOOP SIMULATION USING GAME ENGINE

As stated in the previous section, 17 volunteer drivers were invited to train the NAR neural network in the offline training phase, so the data they generate are considered as "historical data by various drivers" in Fig. 1. However, since the proposed driver behavior modeling methodology is for unknown drivers, we invite another five drivers to test the system in the online actuation phase. Each driver conducts eight simulation trips, so a total of 40 runs are recorded for evaluation.[1]

Two out of those 40 speed trajectories generated by human-in-the-loop simulation runs are selected to conduct an illustrative comparison in Fig. 7, and a better tracking of the advisory speed is observed after implementing the proposed model in general. Note for all speed trajectory figures in this study including Fig. 7(a), (b), Fig. 9(a) and (b): At the same time step, the advisory speed is first generated, the compensated advisory speed is the second (if there is one), and the actual speed is the last. They are not generated at the same time, as illustrated in the system workflow in Fig. 1.

As shown in Fig. 7(a), there is a large speed difference (which indicates a poor speed tracking behavior) at the beginning of the 30-second simulation run, when there is no error prediction model. During this whole run, the red solid line and the dark blue dashed line are not well aligned with each other, indicating that the actual speed generated by the driver deviates from the advisory speed generated by Eq. (2) all the time.

However, as shown in Fig. 7(b), the light blue dotted line denotes the compensated advisory speed calculated by Eq. (8), which predicts driver behavior based on his/her previous driving inputs. For example, at time 7 s, the actual speed (23 m/s) is lower than the advisory speed (21 m/s), so the speed tracking error is -2 m/s. This value along with the values of two previous time steps are the time series inputs of the neural network. The neural network then outputs the prediction speed tracking error at the next time step, which is -3 m/s. This predicted error is added to the advisory speed at time 8 s, so the compensated advisory speed at 8 s is $(20-3=)$ 17 m/s. With the help of the compensated advisory speed, the speed errors are shown to be attenuated during 6–10 s.

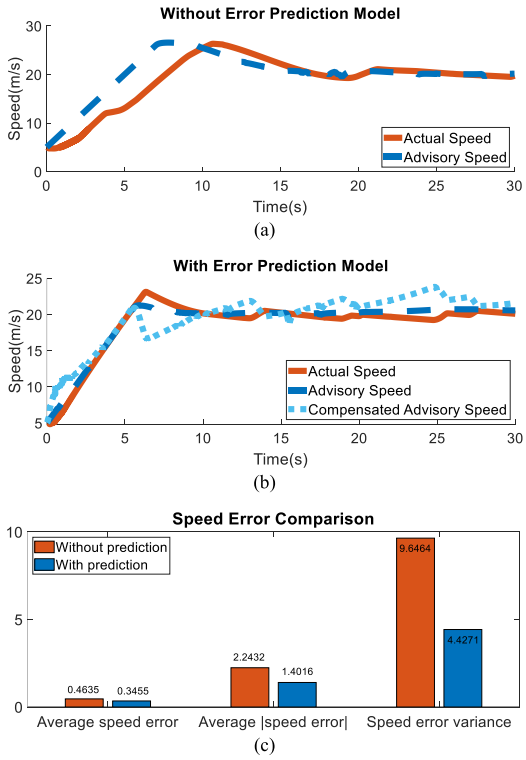


Fig. 7. Speed error comparison in the game engine-based simulation.

Fig. 8. A driver is tracking ASA on HMI in a real passenger vehicle.

[1]The human-in-the-loop simulations are filmed and uploaded to the Internet, where the HUD-based ASA design in Unity can be watched at https://youtu.be/RtrBonGGobg, while the testing of the driving simulator platform (where the data are not used for training and evaluation) can be watched at https://youtu.be/ZgJ_VGuvvz4.

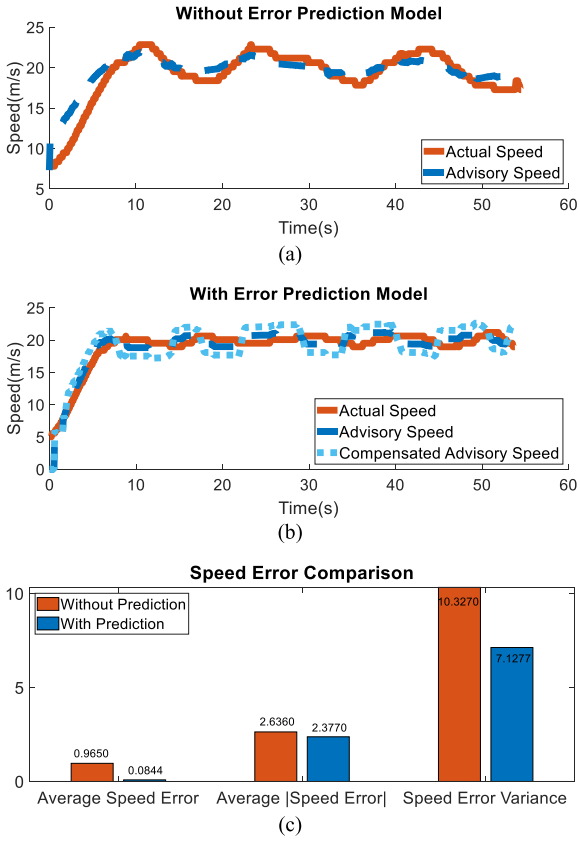


Fig. 9. Speed error comparison in the real vehicle implementation.

In general, with the compensated advisory speed calculated by the error prediction model, the driver can track the advisory speed more precisely than without it, since the red line and the dark blue line are generally closer and less fluctuated in Fig. 7(b) than Fig. 7(a). As shown in Fig. 1, the compensated advisory speed is only adopted in the loop for display purpose, where the speed tracking errors are still calculated by the difference between the actual speed and the advisory speed.

As for the quantitative comparison, we evaluate three different indexes for all 40 runs (with and without the error prediction model) conducted by five drivers. Those three indexes include the mean speed error μ_{δ_v}, the mean value of the absolute speed error $|\mu_{\delta_v}|$, as well as the variance of the speed error σ_{δ_v}. As shown in Fig. 7(c), the mean speed error benefits the least from implementing the error prediction model compared to the other two indexes, with a 23.4% reduction of this index. However, if we take an absolute value of the speed error first, a 36.2% reduction of the index can be observed, which outperforms the previous one. The underlying reason is that, the absolute calculation filters out the situations when the speed errors are bouncing up and down, and positive values offset negative values so the mean values turn out to be relatively small.

As a matter of fact, the speed error variance results in Fig. 7(c) prove the effectiveness of the proposed driver behavior model in an even better way. As shown in the results, the speed error variance is 9.6464 before the error prediction model is implemented, and is cut by half to 4.5661 after the implementation.

TABLE II
ENERGY CONSUMPTION AND POLLUTANT EMISSION RESULTS OF HUMAN-IN-THE-LOOP SIMULATION (ALL VALUES ARE ON A KILOMETER BASIS)

	CO (g)	HC (g)	NO_X (g)	CO_2 (g)	Energy (KJ)
Baseline	2.54	0.0175	0.05721	275.1	3867
Proposed	2.43	0.0167	0.05359	268.2	3770
Reduction	4.3%	4.6%	6.3%	2.5%	2.5%

This 52.7% drop in speed error variance shows that drivers are capable of tracking the advisory speed more closely after the driver behavior model is implemented.

Additionally, we also utilize the U.S. Environmental Protection Agency's MOtor Vehicle Emission Simulator (MOVES) model to perform analysis on the environmental impacts of the proposed model based on all human-in-the-loop simulation runs [43]. As can be seen from TABLE II, the pollutant emissions can be reduced by up to 6.3% after implementing the driver behavior model, and the energy consumption can be reduced by 2.5%, respectively.

VI. REAL VEHICLE FIELD IMPLEMENTATION

In order to validate the proposed game engine-based driver behavior modeling approach, we adopt a real passenger vehicle with automatic transmission to conduct the real-world implementation, which is shown in Fig. 8. The vehicle is equipped with a Windows laptop (processor Intel Core i5-7200U @ 2.50 GHz, 16.0 GB memory) that has access to OBD-II port messages, and also wirelessly connects to an Android tablet which displays ASA to the driver. HMI is designed as Fig. 8, showing the current speed (left) and advisory speed (right) to the driver. This HMI-based ASA is the counterpart of the HUD-based ASA from the game engine-based simulation.

Although we still perform the same cooperative merging scenario described as Fig. 2, the speed trajectory of the main-line leading vehicle is virtually generated for the cooperative merging maneuvers to minimize the system uncertainties. The driver only needs to focus on tracking the advisory speed shown on the HMI-based ASA and validates the effectiveness of the proposed error prediction model. The parameters of the real vehicle implementation are set the same as the human-in-the-loop simulation, which are shown in Table III.

Two out of the ten speed trajectories of the real vehicle implementation are selected to conduct the illustrative comparison, which are shown in Fig. 9(a) and (b), respectively. Coinciding with the simulation result, without the error prediction model, a relatively large gap between the advisory speed and actual speed can be observed during the 60-second test run in Fig. 9(a). However, with the compensated advisory speed calculated by the error prediction model, the driver can track the advisory speed more precisely in Fig. 9(b). Similar to Fig. 7(b), the light blue dotted line in Fig. 9(b) illustrates the compensated advisory speed calculated by Eq. (8), which successfully predicts driver behavior based on his/her previous driving inputs. The red line

TABLE III
PARAMETER SETUP OF THE REAL VEHICLE VALIDATION

Parameters	Host vehicle	Virtual vehicle
GPS antenna to front bumper	2 m	-
GPS antenna to rear bumper	2.9 m	-
Initial speed	4.4 m/s (10 mph)	20 m/s (45 mph)
Advisory speed	-	20 m/s (45 mph)
Desired acceleration range	± 2 m/s^2	0
Speed limit for advisory speed	24.4 m/s (55 mph)	-
Initial intervehicle distance	20 m	
Initial time gap	4 s	
Desired time gap	0.5 s	
Control gains Speed	5	
Control gains Distance	1	
Minimum intervehicle gap	2 m	
Time duration	55 to 60 s	
Communication rate	10 Hz	

TABLE IV
ENERGY CONSUMPTION AND POLLUTANT EMISSION RESULTS OF REAL VEHICLE VALIDATION (ALL VALUES ARE ON A KILOMETER BASIS)

	CO (g)	HC (g)	NO_X (g)	CO_2 (g)	Energy (KJ)
Baseline	1.65	0.01424	0.0481	256.5	3605
Proposed	1.51	0.01192	0.0399	237.9	3344
Reduction	8.5%	16.3%	17.0%	7.3%	7.2%

and the dark blue line are generally closer and less fluctuating in Fig. 9(b) than in Fig. 9(a).

Fig. 9(c) shows that the values of the three indexes are reduced by implementing the error prediction model for all ten runs, which means the real-world implementation validates the results of our game engine-based simulation. Specifically, after compensating the speed tracking errors, a reduction of 20.0% in terms of the speed error variance, and a reduction of 3.5% in terms of the absolute value of average speed error can be obtained, respectively, compared to the baseline scenario when no guidance is provided.

Table IV shows the environmental impacts of the proposed driver behavior model in the real vehicle implementation after implementing the MOVES model. A reduction up to 16.3% in pollutant emissions and a reduction of 7.2% in energy consumption can be obtained after implementing the proposed model, respectively.

VII. CONCLUSIONS

In this study, an ASA system was implemented in the Unity game engine to show an advisory speed to the driver of an intelligent vehicle. A learning-based approach was proposed to predict and compensate for the speed tracking errors generated by different drivers. The effectiveness of the proposed approach in predicting and compensating for the speed tracking errors was validated by the human-in-the-loop game engine-based simulation, which showed a 53% reduction in speed error variance and a 3% reduction in energy consumption, respectively. A real-world implementation with a real passenger vehicle further confirmed the performance of the proposed system.

The proposed driver behavior model is shown to improve the speed tracking capabilities of drivers in both game engine-based simulation and real vehicle implementation. However, based on the quantitative speed error results in Fig. 7(c) and Fig. 9(c), as well as the environmental results in Table II and Table IV, the extents of improvement in these two platforms are quite different. Several potential factors might lead to such differences:

- Vehicle dynamics and powertrain models are far more complex in the real vehicle than what are modeled in the game engine.
- Game engine-based scenarios are well defined, while real-world implementations have more uncertainties, even if they share the same parameter settings.
- Groups of volunteer drivers for the game engine-based simulation and the real-world implementation are not identical, which may cause some biases.

A major future step of this study is to invite more volunteer drivers to conduct more test runs on the driving simulator platform as well as the real passenger vehicle, since the precision of the proposed learning-based approach is heavily dependent on the amount of training data. Another extension of this study is to implement the proposed ASA to other traffic scenarios besides cooperative ramp merging, such as eco-approach and departure, speed harmonization, etc.

ACKNOWLEDGMENT

The authors would like to thanks Dr. Xuewei Qi and Dr. Peng Wang for providing their insights, and all the volunteers at the University of California, Riverside for their contributions in the human-in-the-loop simulation and real-world implementation. The contents of this paper only reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views of Toyota Motor North America, InfoTech Labs.

REFERENCES

[1] J. Piao and M. McDonald, "Advanced driver assistance systems from autonomous to cooperative approach," *Transp. Rev.*, vol. 28, no. 5, pp. 659–684, 2008.
[2] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 7, pp. 1239–1258, Jul. 2010.
[3] G. Hegeman, K. Brookhuis, and S. Hoogendoorn, "Opportunities of advanced driver assistance systems towards overtaking," *Eur. J. Trans. Infrastruct. Res. EJTIR*, vol. 5, no. 4, pp. 281–296, 2005.
[4] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 3, pp. 143–153, Sep. 2003.
[5] N. Van Nes, M. Houtenbos, and I. Van Schagen, "Improving speed behaviour: The potential of in-car speed assistance and speed limit credibility," *IET Intell. Transp. Syst.*, vol. 2, no. 4, pp. 323–330, 2008.
[6] Z. Wang *et al.*, "Early findings from field trials of heavy-duty truck connected eco-driving system," in *Proc. 22th Int. IEEE Conf. Intell. Transp. Syst.*, 2019, pp. 3037–3042.
[7] P. Hao, K. Boriboonsomsin, C. Wang, G. Wu, and M. J. Barth, "Connected eco-approach and departure (EAD) system for diesel trucks," in *Proc. Transp. Res. Board 97th Annu. Meet.*, 2018, pp. 1–6.
[8] X. Qi, P. Wang, G. Wu, K. Boriboonsomsin, and M. J. Barth, "Connected cooperative ecodriving system considering human driver error," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 8, pp. 2721–2733, Aug. 2018.
[9] C. M. Martinez, M. Heucke, F. Wang, B. Gao, and D. Cao, "Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 3, pp. 666–676, Mar. 2018.

[10] Y. L. Murphey, R. Milton, and L. Kiliaris, "Driver's style classification using jerk analysis," in *Proc. IEEE Workshop Comput. Intell. Veh. Vehicular Syst.*, Mar. 2009, pp. 23–28.
[11] V. Butakov and P. Ioannou, "Personalized driver assistance for signalized intersections using V2I communication," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 7, pp. 1910–1919, Jul. 2016.
[12] S. Schnelle, J. Wang, H. Su, and R. Jagacinski, "A driver steering model with personalized desired path generation," *IEEE Trans. Syst. Man, Cybernetics: Syst.*, vol. 47, no. 1, pp. 111–120, Jan. 2017.
[13] Z. Constantinescu, C. Marinoiu, and M. Vladoiu, "Driving style analysis using data mining techniques," *Int. J. Comput. Commun. Control*, vol. 5, no. 5, pp. 654–663, 2010.
[14] C. Miyajima *et al.*, "Driver modeling based on driving behavior and its evaluation in driver identification," *Proc. IEEE*, vol. 95, no. 2, pp. 427–437, 2007.
[15] V. Butakov and P. Ioannou, "Personalized driver/vehicle lane change models for ADAS," *IEEE Trans. Vehicular Technol.*, vol. 64, no. 10, pp. 4422–4431, Oct. 2015.
[16] J. C. McCall and M. M. Trivedi, "Driver behavior and situation aware brake assistance for intelligent vehicles," *Proc. IEEE*, vol. 95, no. 2, pp. 374–387, Feb. 2007.
[17] A. Mudgal, S. Hallmark, A. Carriquiry, and K. Gkritza, "Driving behavior at a roundabout: A hierarchical bayesian regression analysis," *Transp. Res. Part D: Transp. Environ.*, vol. 26, pp. 20–26, 2014.
[18] L. Xu, J. Hu, H. Jiang, and W. Meng, "Establishing style-oriented driver models by imitating human driving behaviors," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2522–2530, Oct. 2015.
[19] A. Augustynowicz, "Preliminary classification of driving style with objective rank method," *Int. J. Automotive Technol.*, vol. 10, no. 5, pp. 607–610, 2009.
[20] Z. Wei, C. Wang, P. Hao, and M. J. Barth, "Vision-based lane-changing behavior detection using deep residual neural network," in *Proc. 22th Int. IEEE Conf. Intell. Transp. Syst.*, 2019, pp. 1741–1748.
[21] C. Guardiola, B. Pla, D. B. Rodrıguez, and A. Reig, "Modelling driving behaviour and its impact on the energy management problem in hybrid electric vehicles," *Int. J. Comput. Math.*, vol. 91, no. 1, pp. 147–156, 2014.
[22] X. Liu, A. Goldsmith, S. S. Mahal, and J. K. Hedrick, "Effects of communication delay on string stability in vehicle platoons," in *Proc. Int. IEEE Conf. Intell. Transp. Syst.*, 2001, pp. 625–630.
[23] S. Lefevre, A. Carvalho, and F. Borrelli, "A learning-based framework for velocity control in autonomous driving," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 1, pp. 32–42, Jan. 2016.
[24] W. Wang, J. Xi, and D. Zhao, "Learning and inferring a driver's braking action in car-following scenarios," *IEEE Trans.Veh. Technol.*, vol. 67, no. 5, pp. 3887–3899, May 2018.
[25] W. Wang, D. Zhao, W. Han, and J. Xi, "A learning-based approach for lane departure warning systems with a personalized driver model," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9145–9157, Oct. 2018.
[26] Z. Wang *et al.*, "Cooperative ramp merging system: Agent-based modeling and simulation using game engine," *SAE Int. J. Connected Autom.Veh.*, vol. 2, no. 2, 2019.
[27] M. Yamaura, N. Arechiga, S. Shiraishi, S. Eisele, J. Hite, S. Neema, J. Scott, and T. Bapty, "ADAS virtual prototyping using modelica and unity co-simulation via openmeta," in *Proc. 1st Japanese Modelica Conf.*, Tokyo, Japan, May 23-24, 2016, pp. 43–49.
[28] B. Kim, Y. Kashiba, S. Dai, and S. Shiraishi, "Testing autonomous vehicle software in the virtual prototyping environment," *IEEE Embedded Syst. Lett.*, vol. 9, no. 1, pp. 5–8, Mar. 2017.
[29] LG Electronics America R&D Center, "LGSVL simulator," 2019-04-28. [Online]. Available: https://www.lgsvlsimulator.com
[30] Unity, "Unity for all," 2020-02-29. [Online]. Available: https://www.unrealengine.com
[31] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," *arXiv:1711.03938*, 2017.
[32] Epic Games, "Unreal engine," 2019-04-28. [Online]. Available: https://www.unrealengine.com
[33] Tass International, "PreScan," 2019-08-15. [Online]. Available: https://tass.plm.automation.siemens.com/prescan
[34] Mechanical Simulation, "CarSim," 2019-08-15. [Online]. Available: https://www.carsim.com/
[35] J. Ma *et al.*, "Freeway speed harmonization," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 78–89, Mar. 2016.
[36] B. Xu *et al.*, "Distributed conflict-free cooperation for multiple connected vehicles at unsignalized intersections," *Transp. Res. Part C: Emer. Technol.*, vol. 93, pp. 322–334, 2018.
[37] Z. Wang, K. Han, B. Kim, G. Wu, and M. J. Barth, "Lookup table-based consensus algorithm for real-time longitudinal motion control of connected and automated vehicles," in in *Proc. Amer. Control Conf.*, pp. 5298–5303, 2019.
[38] Z. Wang, G. Wu, and M. Barth, "Distributed consensus-based cooperative highway on-ramp merging using V2X communications," in *SAE Tech. Paper*, Apr. 2018, pp. 1–10. [Online]. Available: https://doi.org/10.4271/2018-01-1177
[39] B.-W. Yang, "Head-up display for automobile," Mar. 1997, US Patent 5615023.
[40] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Upper Saddle River, NJ USA: Prentice-Hall, 1994.
[41] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: Prediction and system modelling," LA-UR-87-2662; CONF-8706130-4; ON: DE88006479, Tech. Rep., 1987.
[42] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.
[43] U.S. Environmental Protection Agency, "MOVES2014a User Guide," Nov. 2015.

Ziran Wang (Member, IEEE) received the Ph.D. degree in mechanical engineering from the University of California at Riverside in 2019, and the B.E. degree in mechanical engineering and automation from Beijing University of Posts and Telecommunications in 2015, respectively. He is currently a Research Scientist at Toyota Motor North America R&D, InfoTech Labs. His research focuses on cooperative automation and digital twin of intelligent vehicles.

Xishun Liao (Student Member, IEEE) received the B.E. degree in mechanical engineering and automation from the Beijing University of Posts and Telecommunications in 2016, and the M.Eng. degree in mechanical engineering from University of Maryland, College Park in 2018. He is currently a Ph.D. Student in electrical and computer engineering atthe University of California at Riverside. His research focuses on connected and automated vehicle technology.

Chao Wang (Student Member, IEEE) received the B.S. degree in electrical engineering from the Huazhong University of Science and Technology in 2015. He is currently a Ph.D. Candidate in electrical engineering at the University of California at Riverside. His research interests include urban computing, crowdsourced transportation data mining, sharing mobility, connected and automated vehicles, machine learning and deep learning, eco-approach and departure, signal control and traffic operations.

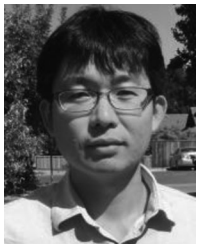
David Oswald received the B.S. degrees in mathematics and computer engineering from the California State University Bakersfield in 2013, and the M.S. degree in electrical engineering from the California State University Northridge in 2017. He is currently pursuing the Ph.D. in electrical engineering from the University of California at Riverside, Riverside. His current research involves connected and automated vehicles.

Guoyuan Wu (Senior Member, IEEE) received the Ph.D. degree in mechanical engineering from the University of California at Berkeley in 2010. He is currently an Associate Research Engineer with the Transportation Systems Research Group, Center for Environmental Research and Technology, Bourns College of Engineering, University of California, Riverside, USA. Dr. His research focuses on intelligent and sustainable transportation system technologies, optimization and control of transportation systems, and traffic simulation.

Kanok Boriboonsimsin (Member, IEEE) received the Ph.D. degree in transportation engineering from the University of Mississippi in 2004. He is currently a Research Engineer with the Center for Environmental Research and Technology, College of Engineering, University of California at Riverside. His research interests include sustainable transportation systems and technologies, intelligent transportation systems, traffic simulation, traffic operations, transportation modeling, vehicle emissions modeling, and vehicle activity analysis.

Matthew J. Barth (Fellow, IEEE) received the M.S. and Ph.D degree in electrical and computer engineering from the University of California at Santa Barbara, in 1985 and 1990, respectively. He is currently the Yeager Families Professor with the College of Engineering, University of California at Riverside, USA. He is also serving as the Director for the Center for Environmental Research and Technology. His current research interests include ITS and the environment, transportation/emissions modeling, vehicle activity analysis, advanced navigation techniques, electric vehicle technology, and advanced sensing and control. Dr. Barth has been active in the IEEE Intelligent Transportation System Society for many years, serving as Senior Editor for both the Transactions of ITS and the IEEE TRANSACTIONS ON INTELLIGENT VEHICLES. He served as the IEEE ITSS President for 2014 and 2015 and is currently the IEEE ITSS Vice President for Finance.

Kyungtae Han (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of Texas at Austin in 2006. He is currently a Principal Researcher at Toyota Motor North America, InfoTech Labs. Prior to joining Toyota, Dr. Han was a Research Scientist at Intel Labs, and a Director in Locix Inc. His research interests include cyber-physical systems, connected and automated vehicle technique, and intelligent transportation systems.



BaekGyu Kim (Member, IEEE) received the Ph.D. degree in computer and information science from the University of Pennsylvania in 2015. He is currently a Principal Researcher at Toyota Motor North America, InfoTech Labs. His research interests include modeling, verification, code generation and model-based testing for high-assurance systems.

Prashant Tiwari received the Ph.D. degree in mechanical engineering from Rensselaer Polytechnic Institute in 2004, and the MBA degree from University of Chicago in 2016. He is currently an Executive Director at Toyota Motor North America, InfoTech Labs. Dr. Tiwari is highly active in Automotive Edge Computing Consortium (AECC) and SAE. Prior to joining Toyota, Dr. Tiwari held several leadership positions of increasing responsibilities at GE and UTC Aerospace Systems.