2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Experimental Evaluation of Deep Neural Networks for Vehicle Model Identification

Amira Moualhi[1,2], Maryam Nezami[1], Saleh Mulhem[2], and Georg Schildbach[1]

Abstract— **In light of the rapid evolution of Artificial Intelligence (AI), a growing number of researchers are investigating the use of Artificial Neural Networks (ANNs) to enhance first-principle Vehicle Models (VMs) or potentially replace them altogether. This paper investigates how AI can be used optimally to identify a VM in the context of a specific case study based on a small-scale experimental vehicle. To this end, three different VMs, each based on a distinct approach, are implemented and compared: (1) a Kinematic Vehicle Model (KVM), (2) a Deep Neural Network (DNN) based VM, and (3) a coupled approach of DNN with KVM, namely Improved KVM (IKVM), where the DNN is used to learn any unmodeled errors produced by the KVM. In the context of the DNN-based approaches, four types of DNNs are implemented based on different configurations of layers (fully connected, convolution, and long short-term memory). For DNN training and evaluation, a custom dataset of driving data is created by driving an *F1tenth* model car for around nine and a half hours on an indoor track while recording all motions using a motion tracking system. The experiments examine the VMs based on multiple performance metrics: the sampling period, 12 different scenarios, and the number of prediction steps the VMs are able to regressively predict without receiving updates regarding extrinsic vehicle states before the error grows too large, i.e., above 1 cm. Our findings are that DNN can increase KVM fidelity substantially. The optimal use of VMs, however, depends on the problem parameters and the vehicle states to be predicted.**

I. INTRODUCTION

Autonomous driving, is one of the most steadily growing technologies, with consistent advances towards full autonomy. Safety concerns are one of the most prominent reasons for the slowed progress of autonomous vehicles [1]. Model Predictive Control (MPC) is a control algorithm that, by means of a model, predicts the future behaviour of the system and then generates inputs to control the system. The predictive performance of the MPC compared to classical control approaches has led to MPC taking over many domains, e.g., autonomous driving. Recently, the application of MPC in autonomous driving architectures, e.g., [2], [3], or safe control architectures, e.g., [4], [5], has demonstrated promising results in improving road safety. A significant challenge in implementing MPC lies in the requirement for high-fidelity models. To address this issue, robust MPC techniques have been introduced, allowing for incorporating model uncertainties, e.g., [6], [7]. However, when dealing with extensive uncertainty sets, the practical applicability of these control methods becomes questionable. For this reason, finding methods to improve the models and decrease their uncertainty is a topic of considerable interest.

[1]Institute for Electrical Engineering in Medicine, Universität zu Lübeck, Lübeck, Germany `a.moualhi@uni-luebeck.de`, `maryam.nezami@uni-luebeck.de`, `georg.schildbach@uni-luebeck.de`
[2]Institute of Computer Engineering, Universität zu Lübeck, Lübeck, Germany `saleh.mulhem@uni-luebeck.de`

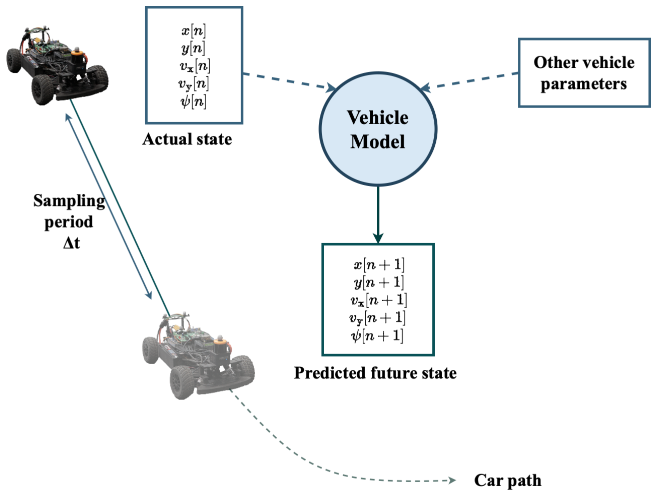

Fig. 1. Prediction of the vehicle's next state.

A. Related Work

Due to the progress made in Artificial Intelligence (AI), there has been a surge of interest in harnessing AI to create high-fidelity Vehicle Models (VMs). AI-based methods can potentially learn intrinsic vehicle parameters based on training data collected by operating the target vehicle platform. Additionally, such methods allow for compensating any unmodeled errors still present in first-principles models. A brief review of these methods is presented in the following.

1) AI for Position Prediction: Finding the next position of a vehicle using Artificial Neural Networks (ANNs) is the centre of interest in some research papers, such as [8], that evaluate Recurrent Neural Networks (RNNs), here in the form of a Long Short-Term Memory (LSTM). In [9], Liu et al. evaluate which input parameters are optimal for ANNs with just a single dense layer, showing that the prediction is no longer useful after two seconds. This paper also indicates that multi-layer perceptrons can perform time series prediction tasks with acceptable results at a time horizon of up to about one second. In [10], an RNN to predict the trajectories of the surrounding vehicle is examined. The idea is to train the RNN on different sampling periods and see how the error varies with long and short sampling period ranges.

2) AI for Acceleration and Yaw Angle Prediction: In [11], Karri et al. use an ANN to predict a vehicle's lateral and longitudinal acceleration and yaw angle based on sensor data. These predictions show sufficient accuracy, with absolute root mean squared error between 10% and 20%.

3) AI for Lateral Dynamics Prediction: In [12], an ANN-based model of a vehicle based on a hybrid learning scheme is proposed. The data used in training is obtained from the actual vehicle measurements. This paper concludes that the ANN-based model is a good approximation. However, it did not consider the longitudinal dynamics of the vehicle.

B. Contributions

The main idea of this paper is depicted in Fig. 1: we investigate three different approaches for system identification and compare their performance in predicting the next state of a vehicle regarding position, velocity and yaw angle. In this study, a custom dataset is experimentally generated in an indoor lab using an F1tenth model car and a motion capture system to record its driving data for around nine and a half hours. The collected data encompasses a large variety of manoeuvres. These data are then used to train different Deep Neural Networks (DNNs) to predict the vehicle's next state (position, velocity and yaw angle). The main contribution of this paper is the design and implementation of various VMs to model an F1tenth car using the generated dataset, where the Kinematic Vehicle Model (KVM) is used as a comparative baseline to our new approaches. For vehicle modelling using DNNs, we distinguish between (1) DNNs used *as* VMs (DNN-based VMs) and (2) a DNN coupled *with* the KVM, which we will refer to as Improved KVM (IKVM). The DNN-based VMs include four different DNN variations, each featuring distinct configurations of fully connected layers, convolutional layers, LSTM layers, and regularization layers such as batch normalization and dropout. To assess the performance of our two approaches (DNN-based VMs and IKVM), we provide extensive testing results based on the collected dataset as well as 12 additional driving scenarios. The performance of our VMs is evaluated on multiple metrics and compared to the baseline KVM.

II. Background

Some nonlinear methods can be applied to accurately approximate the future behaviour of a vehicle, like the dynamic and kinematic models. Each can be realized by either a bicycle or a four-wheel model. The kinematic bicycle model is an accurate, simple way to model the time-dependent behaviour of a vehicle, allowing for predictions [13]. A kinematic bicycle model is a simplified representation of a vehicle derived under the assumption that both front tires are lumped together, both rear tires are lumped together, the tire slip angle in both tires is zero, and there is no rear steering. Under the mentioned assumptions, at time $t \geq 0$ [14], the following differentiable nonlinear continuous-time equations

TABLE I
Vehicle Parameters.

Symbol	Variables	Unit
origin	A point found by the motion capture system as the intersection of the passive markers placed on the F1tenth car	-
x, y	Global x-, y-axis coordinates of the vehicle's origin	m
v_x, v_y	Longitudinal, lateral velocity of the vehicle	m/s
ψ, β	Yaw, side slip angle of the vehicle	rad
l_f, l_r	Distance of the origin to front axle / rear axle	m
δ	Steering angle	rad
a	Acceleration	m/s^2

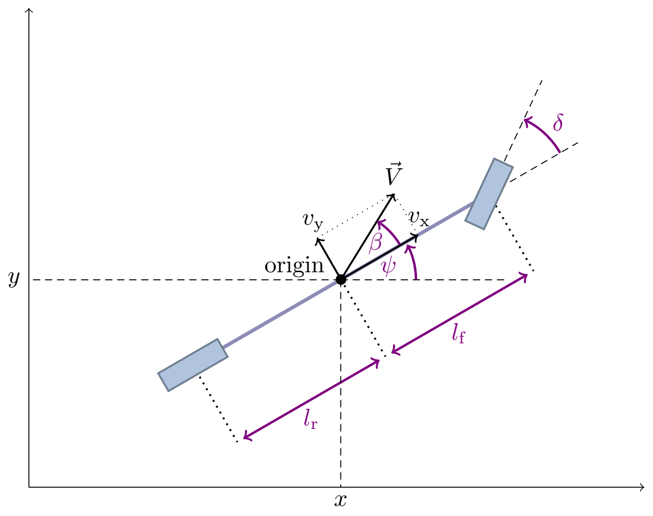

Fig. 2. Illustration of the kinematic bicycle model.

describe the motion of the vehicle:

$$\dot{x}(t) = v_\mathrm{x}(t) = V(t)\cos(\psi(t) + \beta(t)), \quad (1a)$$
$$\dot{y}(t) = v_\mathrm{y}(t) = V(t)\sin(\psi(t) + \beta(t)), \quad (1b)$$
$$\dot{V}(t) = a(t), \quad (1c)$$
$$\dot{\psi}(t) = \frac{V(t)}{l_\mathrm{f} + l_\mathrm{r}}\cos(\beta(t))\tan(\delta(t)), \quad (1d)$$
$$\beta(t) = \psi(t) - \arctan\left(\frac{v_\mathrm{y}(t)}{v_\mathrm{x}(t)}\right), \quad (1e)$$

where x, y and ψ denote the global x- and y-axis coordinates of the origin and the vehicle yaw angle. The origin point is found as the intersection of the passive markers placed on the F1tenth car by the motion capture system, which is explained in the following sections. The vehicle speed is denoted by V, which is the norm of the velocity vector: $V(t) = \sqrt{(v_\mathrm{x}(t))^2 + (v_\mathrm{y}(t))^2}$, where v_x and v_y are the longitudinal and the lateral speed of the vehicle, respectively. The parameter β represents the side slip angle of the vehicle. The vehicle acceleration is denoted as a and the steering

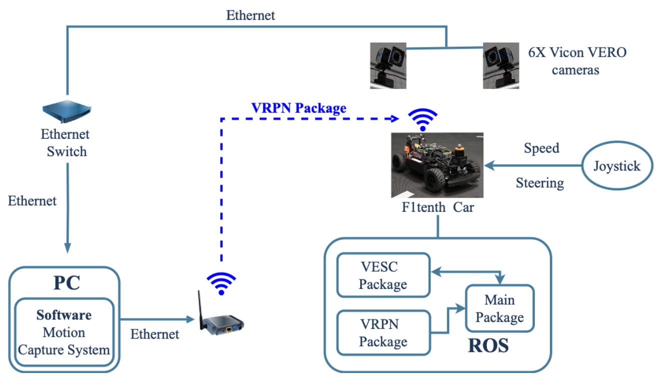

Fig. 3. Connection between the motion capture system and the F1tenth car (adapted and inspired by [21]–[24]).

angle as δ. The model parameters are illustrated in Fig. 2 and presented alongside their units in Table I.

III. Dataset Creation for Vehicle's Next State Prediction

This section is divided into two parts: First, the experimental platform is presented, and second, the process of collecting data is introduced.

A. Experimental Platform

This experimental study is done in an indoor lab, meaning a tracking system is needed to capture the vehicle's motion. Thus, the experimental setup comprises a mobile robot (vehicle) and a motion capture system for data collection. The vehicle runs the Robot Operating System (ROS), allowing all communication to run over ROS messages. The individual parts are illustrated in Fig. 3 and outlined as follows.

1) F1tenth Car as Prototype Vehicle: The prototype vehicle employed in this study is the F1tenth car, an open-source small-scale autonomous vehicle widely utilized in racing competitions [15]. In the F1tenth cars, only the two front wheels can be steered [16]–[19]. The lab observations show that the vehicle's two front wheels have nearly the same steering angle, and for this reason, the following approximations are assumed:

$$\begin{aligned}\delta_{\text{f1}} &= \delta_{\text{f2}} = \delta,\\ \delta_{\text{r1}} &= \delta_{\text{r2}} = 0,\end{aligned}$$

where δ_{f1} and δ_{f2} are right and left front wheel angle and δ_{r1} and δ_{r2} are right and left rear wheel angle, respectively. Also, δ is introduced in (1).

2) Motion Capture System: Six Vicon VERO cameras are used as the motion capture system to track the vehicle's movement in 3 dimensions with 6 degrees of freedom [20] for data collection. The platform setup is illustrated in Fig. 3, where a connection between the motion capture system and the F1tenth car is established.

B. Collecting the Data

Upon establishing the experimental environment, manual control via a joystick is used to drive the vehicle. This approach aims to generate non-ideal driving data, mirroring the nuances of a human driver. A large variety of driving scenarios are represented in the dataset, including driving in all possible directions (forward, backwards, steering right and left), at different velocities and with various steering angles to achieve a high degree of generalisation. In addition, to prevent learning an offset, the origin of the motion capture system is placed at different locations on the indoor track. Supervising the motion capture system while driving is essential to ensure that the collected data is optimal for training because it may lose calibration or tracking on the vehicle if one of the markers is suddenly moved. During an experiment, *rosbag record* is used to record all ROS messages in continuous time so that they can be replayed after the experiment and with an arbitrary sampling period. Further, data cleaning is performed after sampling in order to remove the irrelevant data at the beginning and end of each recording while the vehicle is not yet placed within the bounds of the track (positioning the vehicle manually, not with the joystick). The collected data is not submitted to any other filtering processes since the goal of this study is for the DNNs to learn the compensation of other errors. For data labelling, shifting every input leads to the output for the next state: Output[n]= Input[n+1].

IV. Vehicle Models

In this section, we first introduce the KVM and then present our DNN-based VMs as a substitute. Finally, we introduce our coupled VM: IKVM.

A. Kinematic Vehicle Model

The implementation of the KVM is depicted in Fig. 4. However, it should be noted that the value of the steering command from the joystick is not directly equal to δ in (1), where the constraints on δ are $-15^\circ \leq \delta \leq 15^\circ$ and the joystick steering commands can range from -0.34 to 0.34. To convert the joystick steering command to δ used in (1d), the value of δ is derived experimentally based on lab observations as follows,

$$\delta[n] \approx 44.1^\circ \cdot \text{steer}[n]. \tag{2}$$

Where steer refers to the steering command from the joystick. The next state in the KVM, as presented in Fig. 4, is calculated using the Euler method:

$$Z[n+1] = Z[n] + \dot{Z}[n]\Delta t, \tag{3}$$

where Δt is the sampling period, $Z[n]$ is the available current state of the vehicle from the motion capture system, with $Z[n] = \begin{bmatrix} x[n] & y[n] & v_{\text{x}}[n] & v_{\text{y}}[n] & \psi[n] \end{bmatrix}^\top$, and $\dot{Z}[n] = \begin{bmatrix} \dot{x}[n] & \dot{y}[n] & \dot{v}_{\text{x}}[n] & \dot{v}_{\text{y}}[n] & \dot{\psi}[n] \end{bmatrix}^\top$ is the vehicle states' rate of change, with all entries of $\dot{Z}[n]$ obtained from the motion capture system, except for $\dot{\psi}[n]$ computed using (1d) and (1e).

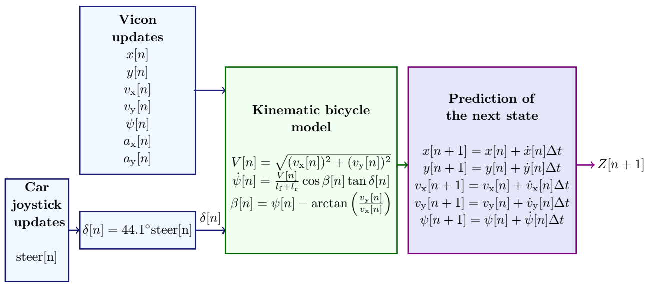
Vicon updates
$x[n]$
$y[n]$
$v_x[n]$
$v_y[n]$
$\psi[n]$
$a_x[n]$
$a_y[n]$
Car joystick updates
steer[n]
$\delta[n] = 44.1°\text{steer}[n]$
$\delta[n]$
Kinematic bicycle model
$V[n] = \sqrt{(v_x[n])^2 + (v_y[n])^2}$
$\dot{\psi}[n] = \frac{V[n]}{l_f + l_r}\cos\beta[n]\tan\delta[n]$
$\beta[n] = \psi[n] - \arctan\left(\frac{v_y[n]}{v_x[n]}\right)$
Prediction of the next state
$x[n+1] = x[n] + \dot{x}[n]\Delta t$
$y[n+1] = y[n] + \dot{y}[n]\Delta t$
$v_x[n+1] = v_x[n] + \dot{v}_x[n]\Delta t$
$v_y[n+1] = v_y[n] + \dot{v}_y[n]\Delta t$
$\psi[n+1] = \psi[n] + \dot{\psi}[n]\Delta t$
$Z[n+1]$

Fig. 4. Kinematic Vehicle Model (KVM).

B. *DNN-Based Vehicle Models*

1) DNN-Based Vehicle Model Architectures: The first VM built is the DNN-based VM. The core idea is to build four types of DNNs, illustrated in Fig. 5, to figure out which type is most appropriate to the time series forecasting problem. Therefore, four DNNs are built and optimised to maximise their performance: two types of DNN (feed-forward and RNN); each type includes two variations (fully connected and Convolution). Convolutional neural networks are compared with MLP to find the most efficient architecture for this forecasting problem. The MLP and CNN consist of fully connected or convolution layers, respectively, while their RNN counterparts add an LSTM layer right before the output layer. It should be noted that the results of the RNNs are obtained in a way that the input state is always treated as the first state to keep the comparison fair between all the VMs. In other words, the hidden states are set to 0; this corresponds to the worst-case for LSTM-based DNNs. This is due to the collected data being from multiple different and independent runs.

2) Feature Selection: After sampling, the dataset is large. Feature selection is applied to (i) reduce the complexity of the DNNs, (ii) eliminate noise that irrelevant features would introduce, and (iii) prevent over-fitting of the DNNs due to many parameters [25]. The motion capture system produces the vehicle's orientation as a quaternion, where all four components are interlinked. As the vehicle only operates on a plane, it can rotate just around the z-axis, making only the yaw angle relevant. The vehicle is controlled with a joystick, providing just steering and speed commands. This means the vehicle has two controllable degrees of freedom, the velocity and steering angle, that control the three total degrees of freedom, which are the position in x and y and yaw angle. Furthermore, all parameters' angular and z components are irrelevant because the vehicle operates on a plane. As those components are constant over time, they only consist of noise that could diminish the performance of the DNNs. Additionally, the impact of the vehicle's acceleration on DNNs performance is experimentally investigated. The results show that the DNNs perform better when the acceleration is not included as an input. The feature selection results in the input vector being reduced to seven variables. The analysis shows that all seven inputs are relevant and essential, meaning no further reduction is required.

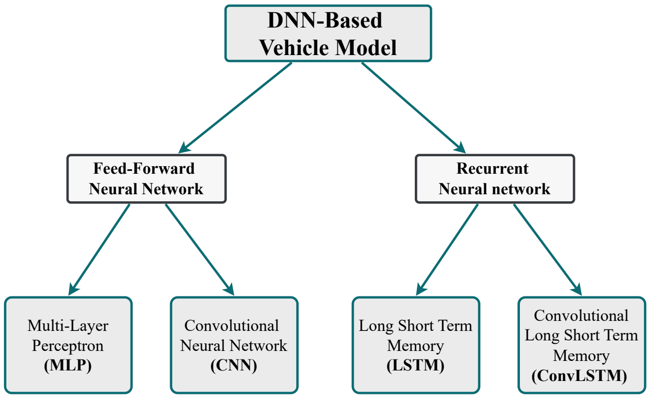
DNN-Based Vehicle Model
Feed-Forward Neural Network
Recurrent Neural network
Multi-Layer Perceptron (MLP)
Convolutional Neural Network (CNN)
Long Short Term Memory (LSTM)
Convolutional Long Short Term Memory (ConvLSTM)

Fig. 5. DNN-based vehicle models.

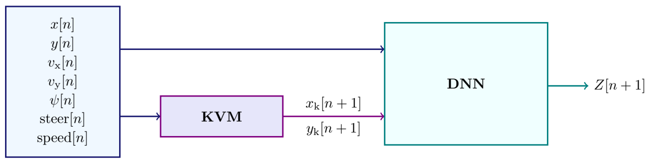

Fig. 6. Coupled KVM with DNN, Improved KVM (IKVM), where $(x_k[n+1], y_k[n+1])$ is the predicted position from the KVM.

C. *Coupling of DNN with KVM: Improved KVM (IKVM)*

For the coupling method, IKVM is illustrated in Fig. 6, the most accurate output from the KVM is used as an additional input to enhance the VM's performance and to predict the unmodeled error that the KVM produces over time. Here, a new DNN architecture needs to be developed as there are no improvements when using the architectures introduced in Section IV-B. On the contrary, the performance of DNN-based VMs deteriorates. This is because they are complex solutions to this problem, meaning a small ANN is needed. For this purpose, the key idea is to work with a shallow neural network. Based on observations while testing the IKVM, working with a shallow neural network with linear activation functions has improved the results of KVM. Furthermore, adding a fully connected layer to the shallow neural network with a *selu* as an activation function improved the IKVM when the steering angle is non-zero (steering scenario).

V. Experimental Results and Comparison

The VMs are evaluated based on three different evaluation criteria: first, the optimal VM regarding the sampling period. Second, the optimal VM regarding scenario evaluation: 12 different combinations of speed and steering and lastly, which VM is more accurate without updating the vehicle's current state. The coefficient of determination (R^2) and D^2 absolute error score (D^2) are used for the evaluation. During the DNNs optimisation process, R^2 shows an almost perfect score for all DNN outputs with insignificant differences, whereas D^2 exhibits a significant margin in performance. This makes room for further improvement. Therefore, the D^2

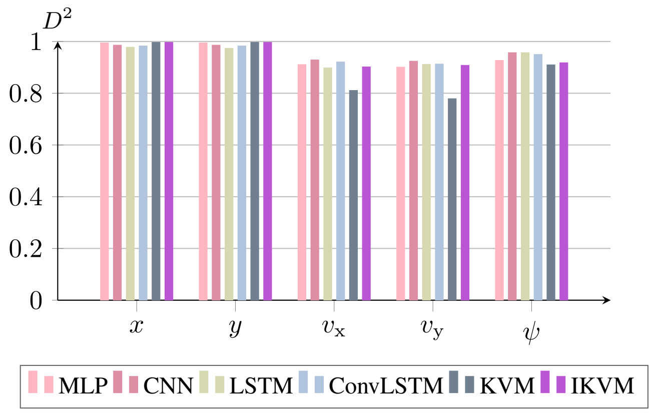

Fig. 7. D^2 for sampling period of 100 ms.

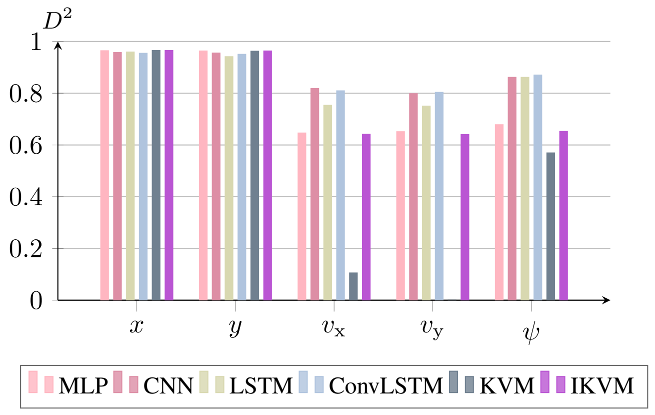

Fig. 9. D^2 for sampling period of 500 ms.

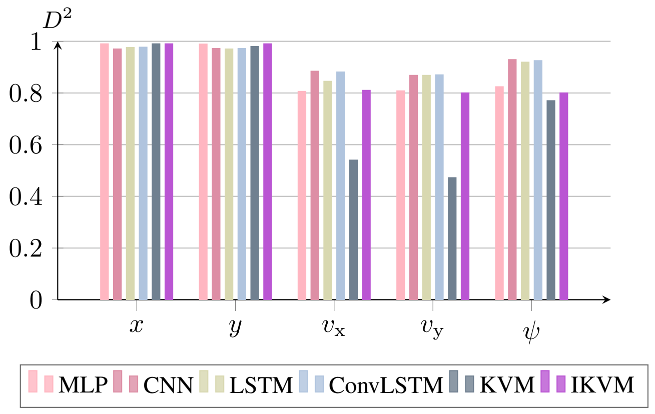

Fig. 8. D^2 for sampling period of 250 ms.

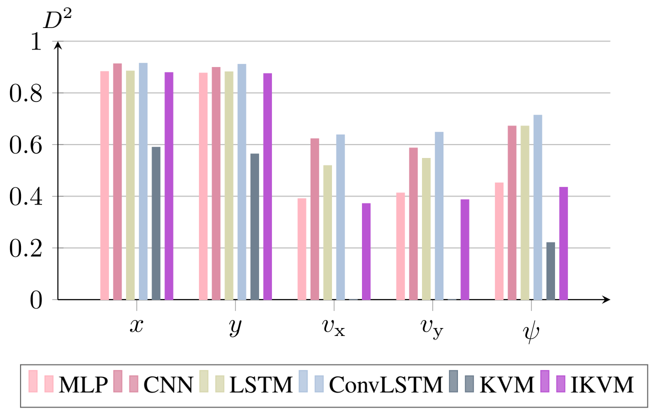

Fig. 10. D^2 for sampling period of 1 s.

score is used to evaluate the following experiments without considering the R^2.

A. Effect of Discretisation on the Vehicle Models

This experiment aims to evaluate all VMs introduced in Section IV (MLP, CNN, LSTM, ConvLSTM, the KVM and the IKVM) to find the optimal VM depending on different sampling periods of (100 ms, 250 ms, 500 ms, 1 s and 2s). Fig. 7 shows that all VMs have a high D^2 score. The IKVM, KVM and MLP have the highest score in terms of predicting the next position ($x[n+1]$, $y[n+1]$), while the KVM has the lowest score when predicting the velocity ($v_x[n+1]$, $v_y[n+1]$) and the orientation (yaw angle: $\psi[n+1]$). The CNN, ConvLSTM and LSTM have the highest score when predicting the yaw angle. In addition, due to the KVM already achieving the maximum possible score regarding the position, the IKVM cannot be further improved. However, when considering the velocity and yaw angle, a significant improvement of IKVM over KVM can be observed at a sampling period of 100 ms. Fig. 8 shows the VMs scoring at a sampling period of 250 ms. The scoring of the predicted velocity from the KVM is lower than 100 ms with 0.2. The DNN-based VMs and the IKVM still offer a better score. However, for predicting the next position, the scoring is higher than 0.97 for all the VMs, where the IKVM, KVM and MLP are still the optimal VMs to predict position. Fig. 9 illustrates how the results change at a sampling period of 500 ms. The v_y score for the KVM is negative, explaining the gap in the plot. This means the KVM can no longer predict velocity. The error-prone current vehicle acceleration provided by the motion capture system has to be used to calculate the next velocity in the KVM, but it is not required by the DNN-based VMs. This explains the KVM's inferior performance in predicting velocity. The IKVM has enhanced all the predictions over the KVM. All the VMs present a high score for predicting the next position. Fig. 10 shows the prediction scores at a sampling period of 1 s. With such a long sampling period, the KVM displays the lowest score, which makes the DNN-based VMs more efficient for all the outputs. It also illustrates that the ConvLSTM becomes less optimal with increasing sampling period due to the information loss caused by the sampling process. Nevertheless, it is the optimal VM to predict the next state vector $Z[n+1]$. The ConvLSTM scores 0.92 at a sampling period of 1 s for predicting the vehicle's position. This also shows that the convolutional

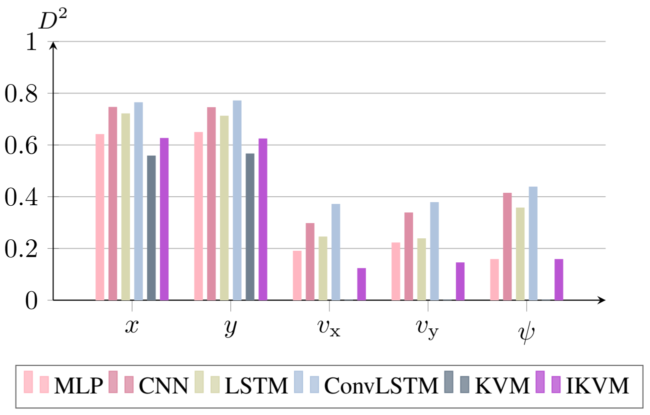

Fig. 11. D^2 for sampling period of 2 s.

neural networks are better than the fully-connected neural networks at a long sampling period. Sampling with a period of 2 s is not practical in real-life situations due to a large amount of information loss and the increased response time. During the sampling period, the joystick values are variable. Therefore, the long sampling period reduces the prediction performance of the VMs. This experiment determines how badly the VMs perform for long sampling periods. Fig. 11 shows that the CNN and ConvLSTM score the highest out of all VMs for a sampling period of 2 s, with ConvLSTM performing best.

In conclusion, at a short sampling period (100 ms, 250 ms and 500 ms), MLP, KVM and IKVM are optimal for predicting the next position, while the DNN-based VMs (CNN, LSTM and ConvLSTM) present the most advantageous VMs for predicting the velocity and orientation. The observations also show that ConvLSTM is the optimal VM with a long sampling period, e.g., 1 s or 2 s.

B. Scenarios Evaluation

In the previous experiment from Section V-A with a short sampling period, some VMs' results are similar. As the test data was collected via random vehicle driving, there might be an imbalance regarding the covered driving scenarios. Therefore, additional experiments are performed to cover specific driving scenarios. Such experiments ensure the decision of the optimal VM for predicting position, velocity and orientation independently from scoring based on test data. Each experiment compares the VMs with similar scoring on predicting the same output. It is evaluated on 12 new and different driving scenarios, which means there are 12 different combinations of steering and speed. This evaluation considers the vehicle moving forward and always steering to the left side due to the results of driving backwards and forward having the same effect. Table II defines the proposed driving scenarios, where $V \approx 1.2\,\mathrm{m/s}$ is the maximum velocity of the F1tenth car. All experiments here are conducted at sampling period Δt = 250 ms. Table III shows the scenario evaluation of the three VMs with the

TABLE II
EVALUATION SCENARIOS.

	w/o steering	**w/ steering**		
Speed	$\delta = 0°$	$\delta \approx 15°$	$\delta \approx 7.5°$	$\delta \approx 3.75°$
$V \approx 1.2\,\mathrm{m/s}$	scenario 1	scenario 4	scenario 7	scenario 10
$V \approx 0.8\,\mathrm{m/s}$	scenario 2	scenario 5	scenario 8	scenario 11
$V \approx 0.3\,\mathrm{m/s}$	scenario 3	scenario 6	scenario 9	scenario 12

TABLE III
OPTIMAL VEHICLE MODEL FOR PREDICTING POSITION.

	Scenarios												
	1	2	3	4	5	6	7	8	9	10	11	12	**Average**
KVM	✓	✓	✓							✓		✓	
IKVM				✓	✓	✓	✓	✓	✓				✓
MLP											✓		

highest scoring when predicting the next vehicle position, where the difference between them is a small variance between 0.01 and 0.001 D^2 score. The KVM is the optimal solution when the vehicle moves without steering. Where the IKVM is the best VM for scenarios 4-9. To summarize this experiment, the IKVM is the most advantageous VM for predicting the position regarding the average score over all scenarios. It is able to correct some errors when they are detected. However, when the error is not detected, the VM's performance is not diminished, thus resulting in a score almost identical to the KVM. The results of the sampling experiment shown in Fig. 8 illustrate that the DNN-based VMs (CNN, LSTM and ConvLSTM) are the optimal VMs for predicting the next velocity and the vehicle's orientation. Table IV clearly shows that CNN is the best VM to predict the velocity ($v_\mathrm{x}[n+1]$, $v_\mathrm{y}[n+1]$), and Table V shows that CNN is the most advantageous VM when predicting the vehicle's orientation. As a result, at a short sampling period, the IKVM is the optimal VM for predicting the position. In contrast, CNN is the optimal VM when predicting the vehicle's orientation and velocity.

C. One-to-Many Evaluation

One-to-many means that all the VMs will get an initial state from the motion capture system and joystick (Input[0]), and after that, it will work only with its predicted output and the updated steering and speed commands from the joystick as the next input, as illustrated in Fig. 12. This experiment aims to uncover which VM performs best for predicting the next position of the vehicle without getting updated on the current state. The evaluation of this experiment is conducted on a random subset of the test data consisting of a steering scenario due to its complexity and low accuracy when predicting the vehicle's position. A sampling period

TABLE IV
Optimal Vehicle Model for Predicting Velocity.

	Scenarios												
	1	2	3	4	5	6	7	8	9	10	11	12	**Average**
CNN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LSTM													
ConvLSTM													

TABLE V
Optimal Vehicle Model for Predicting Yaw Angle ψ.

	Scenarios												
	1	2	3	4	5	6	7	8	9	10	11	12	**Average**
CNN	✓	✓	✓		✓	✓	✓		✓	✓			✓
LSTM													
ConvLSTM				✓				✓			✓	✓	

of Δt = 250 ms is used for this experiment. The KVM has to be adapted to be used for the one-to-many evaluation. Here, the velocity is no longer provided by the motion capture system but rather calculated based on (1a) and (1b). Fig. 13 illustrates the KVM studied in this experiment, using the joystick command and converting it to steering angle and velocity with approximated equations based on lab observation defined as follows:

$$\delta[n] \approx 44.1^\circ \cdot \text{steer}[n]. \tag{4}$$
$$v[n] \approx 0.55 \cdot \text{speed}[n]. \tag{5}$$

Where steer and speed refer to the steering and speed commands from the joystick. Fig. 14 shows the predicted trajectories of the three highest scoring VMs in predicting position (MLP, KVM and IKVM) plotted together with the actual position provided by the motion capture system. Here, the state obtained from the motion capture system serves as the ground truth. The left part of Fig. 14 presents many-to-many (usual case with updating the current state at every time step), showing that the error detected in one step does not grow over time. Unfortunately, this is not the case in the right part of Fig. 14. Here, the error is growing over time. The left part of Fig. 14 shows that the IKVM is the optimal solution, reinforcing the conclusion from the evaluation of the scenarios from the previous experiment. It also shows that the MLP is even better than the KVM. But this is not the case in one-to-many, as evident from the right part of Fig. 14, which shows that the IKVM offers the worst prediction. At the same time, the MLP optimises the position for the next four steps before the error can no longer be controlled. For more precision, the error of every VM produced at each time step is displayed in Fig. 15. The left part of Fig. 15 shows that the three VMs have a slight error in the many-to-many evaluation. For example, the KVM has an error of 1cm in the steering scenario, while the MLP has an error between 0.3cm

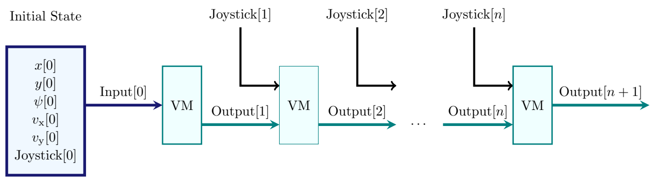

Fig. 12. One-to-many architecture.

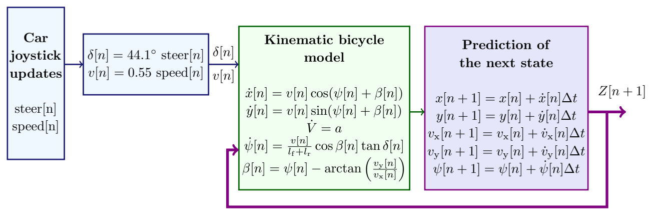
Car joystick updates
steer[n]
speed[n]
$\delta[n] = 44.1^\circ$ steer[n]
$v[n] = 0.55$ speed[n]
$\delta[n]$
$v[n]$
Kinematic bicycle model
$\dot{x}[n] = v[n]\cos(\psi[n] + \beta[n])$
$\dot{y}[n] = v[n]\sin(\psi[n] + \beta[n])$
$\dot{V} = a$
$\dot{\psi}[n] = \frac{v[n]}{l_f + l_r}\cos\beta[n]\tan\delta[n]$
$\beta[n] = \psi[n] - \arctan\left(\frac{v_y[n]}{v_x[n]}\right)$
Prediction of the next state
$x[n+1] = x[n] + \dot{x}[n]\Delta t$
$y[n+1] = y[n] + \dot{y}[n]\Delta t$
$v_x[n+1] = v_x[n] + \dot{v}_x[n]\Delta t$
$v_y[n+1] = v_y[n] + \dot{v}_y[n]\Delta t$
$\psi[n+1] = \psi[n] + \dot{\psi}[n]\Delta t$
$Z[n+1]$

Fig. 13. KVM after adaption to one-to-many.

and 0.7cm. Thus, the IKVM performs best with an error between 0.1cm and 0.5cm. The right part of Fig. 15 shows how the prediction error grows with every new prediction step. It should be noted that the graphic scale has a maximum of 20cm per the IKVM error in the seventh step. At the same time, the MLP outperformed the other VMs by predicting three steps without reaching an error of 1cm. After that, the error gets big, but it remains smaller than the error of the KVM. In summary, the MLP performs best on the one-to-many task.

VI. Conclusion

This paper presented new approaches using DNNs for predicting a vehicle's next state (position, velocity, and orientation). First, we showed how to build a high-quality dataset and multiple optimal DNNs. Then, we performed several experiments. The results showed that the proposed alternative VMs (DNN-based and IKVM) offer better performance than the KVM even at the most critical scenarios (with steering) and at extensive sampling periods of 1 s and 2 s and are thus valid substitutions. We furthermore found that there is no generally optimal VM for predicting the vehicle's next state at all the proposed sampling periods (100 ms, 250 ms, 500 ms, 1 s, 2 s). Instead, which VM is optimal depends on the desired output, reaction time, and the number of prediction steps. For short sampling periods (100 ms, 250 ms, 500 ms), the IKVM outperformed all the other VMs in predicting the vehicle's next position, while the CNN is the optimal VM when predicting orientation and velocity. For sampling periods of 1 s and 2 s, the ConvLSTM is the most advantageous VM regarding all outputs. In Addition, in case of no extrinsic updates to the current state, the MLP can outperform the other VMs at a sampling period of 250 ms by being able to predict three position steps without reaching an error of 1cm.

Acknowledgment

Research leading to these results has received funding partially from the EU Horizon Europe research and innovation

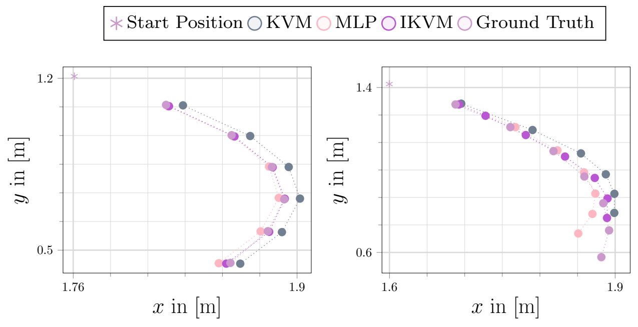

Fig. 14. Vehicle trajectory regarding the predicted steps. Left in case of an update, right in case of no update.

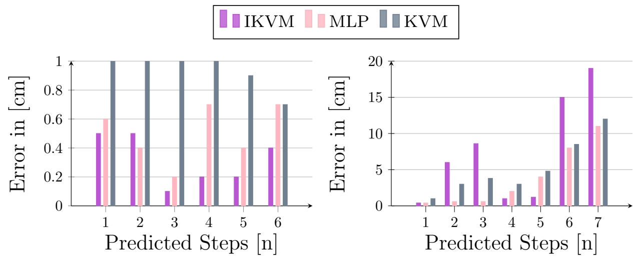

Fig. 15. Position errors regarding the predicted steps. Left in case of an update, right in case of no update.

program, Chips Joint Undertaking, under grant agreement no. 101112089 (project AIMS5.0) and from the partners national funding authorities, the Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF).

References

[1] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, 2020.

[2] R. Kensbock, M. Nezami, and G. Schildbach, "Scenario-based decision-making, planning and control for interaction-aware autonomous driving on highways," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–6.

[3] S. H. Nair, V. Govindarajan, T. Lin, C. Meissen, H. E. Tseng, and F. Borrelli, "Stochastic MPC with multi-modal predictions for traffic intersections," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, 2022, pp. 635–640.

[4] B. Tearle, K. P. Wabersich, A. Carron, and M. N. Zeilinger, "A predictive safety filter for learning-based racing control," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7635–7642, 2021.

[5] M. Nezami, G. Männel, H. S. Abbas, and G. Schildbach, "A safe control architecture based on a model predictive control supervisor for autonomous driving," in *2021 European Control Conference (ECC)*, 2021, pp. 1297–1302.

[6] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[7] M. Nezami, H. S. Abbas, N. T. Nguyen, and G. Schildbach, "Robust tube-based LPV-MPC for autonomous lane keeping," *IFAC-PapersOnLine*, vol. 55, no. 35, pp. 103–108, 2022.

[8] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 353–359.

[9] Q. Liu, B. Lathrop, and V. Butakov, "Vehicle lateral position prediction: A small step towards a comprehensive risk assessment system," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 667–672.

[10] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 399–404.

[11] V. Karri and D. Butler, "Using artificial neural networks to predict vehicle acceleration and yaw angles," in *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, vol. 4, 2002, pp. 1915–1919.

[12] Y. U. Yim and S.-Y. Oh, "Modeling of vehicle dynamics from real vehicle measurements using a neural network with two-stage hybrid learning for accurate long-term prediction," *IEEE Transactions on Vehicular Technology*, vol. 53, no. 4, pp. 1076–1084, 2004.

[13] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, pp. 1094–1099.

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

[15] "F1tenth," https://www.F1tenth.org.

[16] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[17] V. S. Babu and M. Behl, "f1tenth.dev - An open-source ROS based f1/10 autonomous racing simulator," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 1614–1620.

[18] P. Gautam, "Designing variable ackerman steering geometry for formula student race car," 2021.

[19] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *arXiv:2202.07008*, 2022.

[20] "Vicon," https://www.Vicon.com.

[21] Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System." [Online]. Available: https://www.ros.org

[22] C. Jun, J. Y. Lee, B. H. Kim, and S. Do Noh, "Automatized modeling of a human engineering simulation using kinect," *Robotics and Computer-Integrated Manufacturing*, vol. 55, pp. 259–264, 2019.

[23] D. Montufar, F. Munoz, E. Espinoza, O. Garcia, and S. Salazar, "Multi-UAV testbed for aerial manipulation applications," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014, pp. 830–835.

[24] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: a device-independent, network-transparent VR peripheral system," in *Proceedings of the ACM symposium on Virtual reality software and technology*, 2001, pp. 55–61.

[25] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.