

Bridging the Gap between Open Loop Tests and Statistical Validation for Highly Automated Driving

Alexander Koenig and Michael Gutbrod
BMW AG

Sören Hohmann and Julian Ludwig
KIT Karlsruhe Institute of Technology

ABSTRACT

Highly automated driving (HAD) is under rapid development and will be available for customers within the next years. However the evidence that HAD is at least as safe as human driving has still not been produced. The challenge is to drive hundreds of millions of test kilometers without incidents to show that statistically HAD is significantly safer. One approach is to let a HAD function run in parallel with human drivers in customer cars to utilize a fraction of the billions of kilometers driven every year. To guarantee safety, the function under test (FUT) has access to sensors but its output is not executed, which results in an open loop problem. To overcome this shortcoming, the proposed method consists of four steps to close the loop for the FUT. First, sensor data from real driving scenarios is fused in a world model and enhanced by incorporating future time steps into original measurements. Second, recorded time-shifted data is used to identify intentions of each driver of the surrounding traffic. Third, the real scene is used as start scene for simulation. Reactions of surrounding traffic to the FUT output are simulated using identified intentions from step two. Fourth, simulation output is evaluated for erroneous behavior of the FUT to give developers input for further improvement and provide statistics about the safely driven distance. Main contribution is the first complete method for passive HAD safety assessment with a solution to the open loop problem. Results are presented for driver intention identification and criticality assessment.

CITATION: Koenig, A., Gutbrod, M., Hohmann, S., and Ludwig, J., "Bridging the Gap between Open Loop Tests and Statistical Validation for Highly Automated Driving," *SAE Int. J. Trans. Safety* 5(1):2017, doi:10.4271/2017-01-1403.

INTRODUCTION

Development for highly automated driving (HAD) is progressing quickly. Various car manufacturers [1] have announced to offer a self-driving function equivalent to SAE Level 4 [2] for sale within the decade.

Besides better usage of time in the car one of the main arguments for self-driving vehicles are the expected safety gains with the long term goal of accident-free traffic [3]. But showing that automated cars are safer drivers than most humans is not a trivial task, since accidents with damage to persons occur very rarely. On German freeways such an accident happens statistically once every 12 million kilometers [4], a number which is expected to further increase with cars getting safer with every innovation cycle. In order to prove statistically that a self-driving function for freeways is at least as safe as the average human driver, hundreds of millions of kilometers would have to be driven, repeated for every version of the function under test (FUT) [5]. Thus, even though the introduction of self-driving cars is generally expected to save lives by avoiding common human mistakes which are the cause for more than 90% of all accidents [6], it is difficult to actually produce statistical evidence. However,

permitting a system that could result in more accidents than the status quo is hard to justify. Consequently, a scientific proof is considered a necessity for authorities.

Since closed-loop test drives of the described magnitude are neither economically nor by expenditure of time a practical solution, verification by simulation does seem like the straight forward answer. One of the major difficulties of simulation is to show its validity, since only those situations that have been encoded as possibilities into the simulation upfront can be tested. Due to limitations of every simulation environment, this can only be a limited amount of different situations, whereas reality is continuous and unlimited and thus every FUT will be faced with an infinite number of different situations [7].

As a novel approach avoiding drawbacks of both test drives and simulation, an open loop test in customer cars was suggested in [5] and [8], named Trojan Horse. As general problem with open loop tests, the divergence between the real life situation and progress of the hypothetical situation where the FUT has been active is identified by the authors, but no solution for non-stationary parts of scenes are given.

In this work, we decided to enhance the Trojan Horse approach in order to close the open loop for the FUT. We simulate reactions of the dynamic part of the environment to the FUT. To increase validity, we aim at using most of the information the real life situation provides. The assumption is that driver actions change depending on the surrounding traffic whereas their goals and intentions like desired velocity, favorite lane and route remain unchanged and can be identified and modeled.

The proposed approach's main differences to classical simulation is using the real life scene as starting point for simulation and incorporating information derived from the development of the real scenario to increase validity.

The main difference to closed-loop test drives is the open-loop and therefore risk-free and cost-efficient employment of the FUT, which allows it to be applied in any car equipped with the respective sensor and ECU setup.

Furthermore, once recorded scenarios can be resimulated every time a new version of the FUT is released, reusing the knowledge derived earlier and avoiding additional costs of real life tests.

For the remainder of this paper and in accordance to [9] we use the following definitions to facilitate understanding of the method:

- *scenery* - the stationary parts of the environment
- *scene* - all dynamic and stationary parts in one time step
- *scenario* - scenes over time
- *situation* - scenario from a subjective perspective, including only elements relevant for a given function or driver

The rest of the paper is organized as follows: First the overall method is explained in detail, with special regard to driver intention recognition, traffic simulation and criticality assessment of simulated scenarios. Second, results are presented for driver intention recognition with neural networks and criticality assessment with a state machine. Last conclusion and outlook are given.

METHOD

The presented method, for which we will use the term passive HAD, consists of the following steps: 1. Record real traffic scenario. 2. Identify models θ_i of other drivers in scenario. 3. Simulate scenario with output of FUT γ using models θ . 4. Assess simulated scenario for criticality (Figure 1):

If high criticality is given, the scenario can be resimulated with varying θ and γ and afterwards be reassessed for criticality.

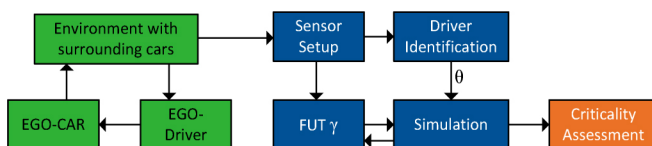


Figure 1. Green: Control loop of human driver. Blue: Control loop of passive HAD. Orange: Criticality Assessment.

Identification of Driver Model θ_i

The main idea behind identifying the surrounding drivers is that goals and intentions are parameters of a driver model and as such traffic independent. Actions on the other hand are the observable output. They result from a particular traffic situation which is used as input to the driver model. By identifying the driver's intentions from a real life situation, we have a specific model to derive how the respective driver would act if a situation was different from the real life situation. This can be used to simulate actions in a scenario different to the original one with regard to the ego vehicles behavior.

Identification of the driver models has to rely on the data provided by sensors, which limits the knowledge that can be acquired due to sensor availability, their range and measurement noise. For this reason, the proposed identification methods are restricted to the following data:

- the current velocity of the vehicle to be identified
- currently valid speed limits
- existence of neighbor lanes
- relative velocities $v_{ego-n} \forall n \in \{\text{predecessor, successor}\}$ of own and neighbor lanes
- relative distances $d_{ego-n} \forall n \in \{\text{predecessor, successor}\}$ of own and neighbor lanes

Additionally, time headway (THW) and time to collision (TTC) are used in most traffic simulation models and are easy to calculate from the above data, so that they are employed in identification of driver intentions and in criticality assessment as well.

THW is proportional to the distance between a leading vehicle L and a following vehicle F, relative to the velocity driven, with $THW = \frac{p_L - p_F}{v_F}$, p for longitudinal position and v for absolute velocity. In other words, THW is the time the following car needs to reach the position of the leading.

TTC depends on the distance between two vehicles relative to their velocity difference, with $TTC = \frac{p_L - p_F}{v_F - v_L}$. Thus, TTC is the time in which a collision will occur if none of the vehicles changes velocity or direction. If the leading vehicle is at least as fast as the following, TTC is infinite or negative.

These features do not perfectly reflect the information a human driver could take into account for deciding which lane to use or which velocity to drive, therefore any identification can only be an approximation of the actual intentions of the driver.

Identification of Driver Intentions with Deep Neural Networks

Deep Neural Networks have proven to be powerful tools used in various areas of research such as visual object recognition, image classification, speech recognition and many other domains [10, 11]. In this work, two different methods are evaluated concerning performance in modelling driver intentions and on their ability to provide additional information alongside the classification result.

We consider the driver intentions as a function f^* , with $y = f^*(x)$, that maps an input vector x to an intention category y . The input vector x contains data that can be captured with state of the art commercial automotive sensors, as explained beforehand.

As one of the most important use cases of HAD will be driving on freeways, the set of driver intentions is discretized into different lane change wishes. It should be noted that this discretization is not applicable to more complex driving situations in e.g. urban areas or roads without lane markings distinguishing individual lanes. In these cases a trajectory based approach in which the neural net predicts exact positions and velocities would be better suited. However, due to additional input features needed, e.g. more road topology information and a wider range of possible maneuvers and hence a higher individuality of drivers, this would be too complex for assessing the HAD functionality as described above.

We show that our approaches are not only able to classify lane change wishes, but also work as prediction method for actual lane changes. Thus we distinguish between two output categories $y_{intention}$ and y_{action} , which are covered in separate training and evaluation phases, but are based on the same data sets.

$$y_{intention} \in \begin{cases} \text{lane change wish left} \\ \text{lane change wish right} \\ \text{keep lane wish} \end{cases}$$

$$y_{action} \in \begin{cases} \text{lane change left} \\ \text{lane change right} \\ \text{keep lane} \end{cases}$$

Data Preprocessing

The provided learning data contains either a “keep lane”-, a “change to right lane”-, or a “change to left lane”-wish, with those wishes being true only in a subset of all time steps within a sequence. The training set is composed in a way that each sequence category represents one third of all sequences, to appropriately address the problem of skewed classes. Furthermore the training set is shuffled before the beginning of the training phase and we draw random batches from it during training. Moreover we used min/max normalization $z_i = \frac{x_i - \min(X)}{\max(X) - \min(X)}$ as data preparation, with z_i being the normalized i^{th} data point from data set X .

Deep Feedforward Network

In order to approximate f^* we applied a deep feedforward network with six hidden layers to our problem domain using the TensorFlow framework [12]. The detailed hyperparameter configuration of the network is provided in Table 1 and is the result of a hyperparameter optimization which compared the performance of different network configurations with each other. The state of the art Adam optimizer [8] and Xavier initializer [13] are used for the deep neural network because of best performance.

Table 1. Structure of deep neural network implementations.

	Deep feedforward network	VAE
Hidden layer	6	2 x 3
Hidden nodes at each layer	200	1 st layer: 200 2 nd layer: 100 3 rd layer: 50
Activation function	ReLU (rectified linear unit)	ReLU
Optimizer	Adam [8]	Adam [8]
Weights initializer	Xavier Init [13]	Xavier Init [13]
Learning rate	0.0003	0.0003
Batch size	1000	1000
Latent variables	-	30

Variational Autoencoder (VAE)

Applying the feedforward network yielded already satisfying classification performance, however the network lacks a possibility to reproduce on what basis classification decisions are made, as it is not focused on proving that results lie in certain confidence intervals [14]. For this reason we deploy a VAE network (Figure 2) that supports such probabilistic considerations.

An autoencoder is an hourglass shaped neural network that attempts to copy its inputs to its outputs. It consists of two main parts, an encoder that maps the inputs to 30 hidden variables (Table 1) called latent space and a decoder that produces a reconstruction out of those latent variables [14]. As an intended consequence of the hourglass shape with fewer neurons in the inner parts than in the outer, autoencoders are unable to copy perfectly from their inputs to their outputs, hence they are forced to prioritize on useful properties of the presented data, which are learned and reflected in the latent space.

Variational Autoencoders (VAE) as proposed in [15, 16] are a specialized class of autoencoders and allow learning probabilistic encoders and decoders of data. Consequently the classification decision of the network gets augmented by a probability information, which reflects how confident the network is about its classification.

Formally speaking, we consider examples X in our datasets distributed according to some unknown distribution $P_{generate}(X)$ while our goal is to learn a model P which we can sample from, with P being as similar as possible to $P_{generate}$ [17, 18]. In contrast to autoencoders VAEs (Figure 2) map inputs x to their expected value $E(x)$ and the variance $Var(x)$ of the underlying distribution $P_{generate}(X)$. VAEs provide the advantages of fast training via backpropagation while requiring only weak assumptions concerning the structure of the data. Nevertheless they introduce a certain approximation error, which is relatively small given high-capacity VAEs.