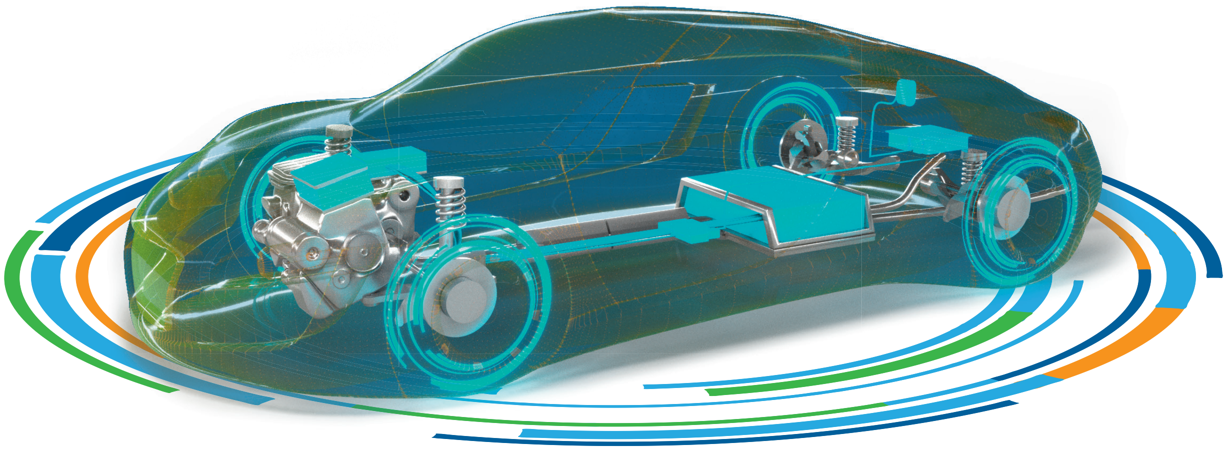


INTERNATIONAL SYMPOSIUM ON DEVELOPMENT METHODOLOGY

Comparison of driver models for powertrain test benches using a digital twin

Jannes Schilling, Dr.-Ing. Jan-Michael Wilmsen, Paul Nitschke – Dr. Ing. h.c. F. Porsche AG;-
Prof. Dr.-Ing. Hans-Christian Reuss – Universität Stuttgart, Stuttgart, Germany

ISBN 978-3-9816971-6-2

Abstract

At the powertrain test bench, all powertrain components from the engine to the side shafts can be tested without a prototype vehicle. With detailed vehicle, driver and environmental models and real-time simulation capabilities, it is possible to run partially virtual and highly realistic tests and applications. Therefore, the validation of the powertrain can be done at lower costs and at an earlier stage under defined, reproducible boundary conditions. As the complexity of the models increases, so does the effort required for parameterization prior to each test run. To shorten the duration of commissioning, previous work has focused on designing a digital twin of a Powertrain-in-the-Loop test bench (vPiL). The vPiL includes powertrain models, enabling virtual parametrization of the simulation models before the test run. Furthermore, the digital twin is also used for the development or integration of new simulation models without generating cost-intensive runtimes at the test bench. This paper focuses on how to properly model a driver for the powertrain test bench which differs from many current driver models, for example for autonomous driving (AD) functions. In AD, the driver is influenced by previously unknown, external factors such as other drivers which impede automation. On the powertrain test bench, however, the entire test scenario is already known in advance. The focus of the driver model is therefore on the most accurate control possible to follow the known trajectory.

In addition, realistic driving behavior is required to avoid unrealistic loads on the powertrain. The main contribution of this paper is an empirical comparison of different driver models. First, we will introduce three driver types which are respectively based on PI-control systems, adaptive PI-control systems, and reinforcement learning. The PI-controller will be parameterized in the vPiL to ensure optimized and operator independent parameterization. The reinforcement learning model is based on the Soft Actor-Critic algorithm. Second, we will motivate the models' advantages and disadvantages regarding their usage on a test bench. Our focus is on the models' accuracy, required computation power as well as setup time and complexity. Finally, we will discuss the technical implementation of all three models in a MathWorks® Simulink model. This Paper will allow test bench engineers to select a matching driver model for partially virtual validation of powertrain components on a powertrain test bench. In addition, potentials of the various models for future development will be identified.

1 Introduction

A powertrain test bench enables testing of internal combustion engine, hybrid, and electric powertrains in different test scenarios. Often, these scenarios are intended to provide comparable results to vehicle tests. One way to replicate vehicle testing on the test bench is to follow existing trajectories. These trajectories consist of a wheel speed trace and a pedal trace. The trajectory is used to control the speed of the dynamometer and the powertrain. However, this method has two drawbacks. First, the dynamics of the powertrain dynamometer may differ from the dynamics of the vehicle, resulting in different loads. Second, other units under test (UUT) may behave differently than the powertrain in the reference vehicle, causing further deviation. A suitable method is the operation of the powertrain test bench in combination with a real-time simulation as a powertrain-in-the-loop (PiL) test bench. In this method, the test is described by road characteristics in form of a speed profile and a curvature profile. The real-time simulation uses an environmental, vehicle, and driver model to calculate setpoint values for the dynamometer and UUT. This ensures that the load applied to the UUT is comparable to the vehicle test.

The real-time simulation driver can influence the vehicle's longitudinal and lateral dynamics by accelerating, decelerating, and steering. In doing so, the aim is to follow the referenced trajectory as accurately as possible. Since the accuracy of the driver affects the overall vehicle dynamics, it is important for the quality of the test results. Therefore, this paper compares widely used control theory-based drivers with a new driver based on reinforcement learning.

The general principles of driver modeling and current research in both areas are described in Chapter 2. Chapter 3 describes the framework and requirements of the drivers, before the control theory-based driver is presented in Chapter 4 and the reinforcement learning driver in Chapter 5. A comparison of the two drivers is given in Chapter 6.

2 Related Work: Driver Modeling for PiL Test Bench

A driver's primary task is to maneuver a vehicle safely from a starting point to a predetermined destination. This can only be achieved through the driver's awareness of the vehicle's environment and the control of the vehicle's actuators. [1] This complex task can be divided into different hierarchies using the 3-level model (Figure 1). At the strategic level, the driver generates a route based on a starting point and the traffic situation. Within the maneuvering level, the route is transformed into a specific maneuver. The selection of the maneuver is based on the vehicle's position and movement, as well as the driving space, and can be optimized according to various criteria. Lastly, the maneuver is executed within the control level using the steering wheel and pedals of the car. [2]

On a PiL test bench, the strategic level is not of interest as the test case defines the route before the test. Whether to model the maneuvering level depends on the test case. To simulate driving scenarios as realistically as possible, some systems include environmental simulation with randomly generated traffic. Other test cases are based on the reproducible execution of various maneuvers. These maneuvers are described by a predetermined trajectory composed of speed and curvature. Consequently, only the implementation of the control level is used on the test bench. Since the number of test cases with a predetermined trajectory outweighs the others, this paper concentrates on the modelling of the control level using both control theory and machine learning approaches.

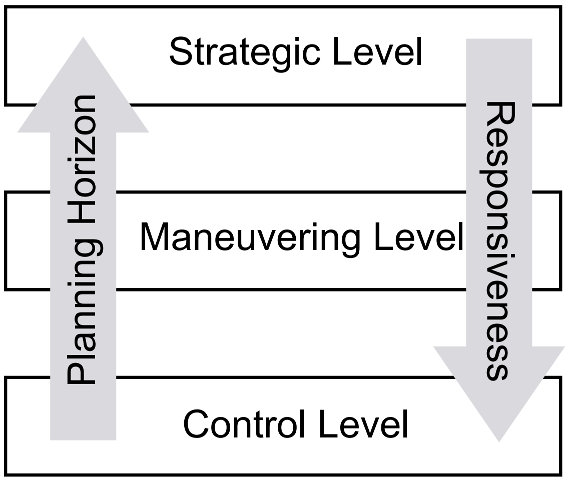


Figure 1: Three-Level-Model of Driving Behavior [3]

Most publications regarding the control level, also known as trajectory-following control, are related to autonomous driving or driver assistance systems. However, these approaches can also be applied to powertrain test benches.

Kebbati et al. introduce a self-adaptive PID controller in [4]. Initially, the authors establish a digital PID controller in a basic vehicle model utilizing an optimization algorithm as a benchmark. The self-adaptive controller uses a neural network to dynamically adjust the controller parameters (K_p,K_i,K_d). This allows the control system to respond to external disturbances such as wind or road gradient. Aziziaghdam presents an approach that combines a PI-controller with an inverted vehicle model in [5]. The controller determines its outputs by comparing the target speed with the current speed of the vehicle. Subsequently, the setpoint value is converted into an accelerator setpoint along an acceleration path or into a brake setpoint along a deceleration path, depending on the vehicle's acceleration. The two paths feature an inverted model of the vehicle's braking and acceleration paths. In contrast with the prior model, creating the inverted model necessitates knowledge of the vehicle parameters (engine map, overall gear ratio etc.). In [6], the driver model combines a vehicle model as a feedforward with a PI-Controller. By incorporating foresight with future velocity values, the state of rolling or sailing can be assumed in addition to braking and accelerating.

Reinforcement Learning (RL) for large, continuous environments is a relatively young research area. Nonetheless, there has been initial work on the usage of RL for longitudinal control. Buechel introduces in [7] a RL-based controller that converts current and future reference speeds into accelerator and brake setpoints. While this approach achieves a low error rate, the paper does not discuss how to ensure realistic driver behavior. Wang [8] describes a RL-based controller for an adaptive cruise control system that consists of two levels. The first level contains the RL-based controller which converts the speed and distance difference to the next vehicle into a target acceleration. The second level consists of physical models for the acceleration and delay path which convert the target acceleration into accelerator and brake setpoints.

3 Setup and Requirements

This chapter presents an analysis of the PiL test bench with a focus on the driver subsystem. The interfaces to the other subsystems, the requirements for the driver model, as well as the applicability of data-based methods are of interest.

3.1 System Analysis

The test bench's real-time simulation has three linked subsystems: the environmental model, the vehicle model, and the driver model. This virtual system interacts with two physical subsystems: the dynamometers and the attached UUT.

The real-time simulation receives an input of trajectory data, including information on the test track and test boundary conditions. Both the environmental model and the driver model have access to this information. The system's second input is the actual speed of the dynamometers, which corresponds to the wheel speed of the UUT. The real-time simulation generates torque setpoints for the dynamometers, and an accelerator setpoint, which is transmitted to the UUT.

The driver model's inputs are comprised of the speed and curvature trace of the trajectory, as well as the current speed of the vehicle, which is calculated by the vehicle model. In addition to the accelerator, the driver model generates a brake setpoint and a steering wheel setpoint, both of which have a direct impact on the vehicle model. To simplify the development of the new models, this paper's investigations focus on maneuvers that don't involve lateral dynamics. As a result, the driver model's outputs are reduced to the accelerator and brake.

For the development of the control theory approach, an analysis of the control theory plant is necessary. While having a model of the plant is beneficial, it is not mandatory. Parameterization of the driver model can be performed on the PiL test bench using various experimental programs with the current UUT. The Reinforcement Learning (RL) approach, on the other hand, involves testing various accelerator and brake setpoints to solve an optimization problem. If these experiments are performed in a semi-virtual system such as PiL, they cannot be parallelized and must be performed in real time. The resulting costs and duration make such approaches unappealing. The authors have previously introduced a virtual powertrain-in-the-loop test bench (vPiL) in prior publications [9]. In addition to the real-time simulation, the vPiL also replicates the test bench and the UUT virtually, so that the training of the reinforcement learning approach can take place in the vPiL before the actual test run. A neural network representing the driver model can finally be used in the real PiL test bench.

3.2 Driver Model Requirements

To achieve applicable driver modeling for use on a test bench, it is crucial to first establish specific requirements. Table 1 displays the requirements arranged according to significance. The key role of the driver model is to accurately control the reference vehicle speed. Also, the controller's actuating variables should mimic those of a human driver. This means: The driver model should not use the accelerator and brake pedals at the same time or switch between them high-frequently. This can be tracked by the number of transitions between the two states acceleration and deceleration. Moreover, the computational demands of the model should not be overly burdensome, as it must run in real-time simulation within the test bench automation system. The driver model must meet the initial three criteria sufficiently to be utilized in test bench operations.

The remaining requirements are of lesser importance but increase the model's manageability. The driver model should be able to operate independently of vehicle or track type. For instance, the vehicle dynamics can vary due to weight or power, while the trajectories may differ due to varying speed or speed gradient profiles. Despite the driver model's complexity, the parameterization effort should be minimal. On one hand, to reduce the time required for driver model setup; on the other hand, to obtain better comparability between different test runs. The requirement Explainability pertains to the comprehensibility of the model. A self-explanatory model supports analysis and debugging, especially in the case of an error. Furthermore, different driving habits, such as sporty or economical, are interesting, but they always involve a change in vehicle speed, which contradicts the initial requirement. Therefore, the representation of the driving styles is a part of the maneuver level and will not be considered any further here.

	Requirements
1	Minimum deviation from the reference trajectory
2	Realistic actuating setpoints
3	Low computing effort
4	Independent of vehicle and track type
5	Low parameterisation effort
6	Explainability
7	(Various driving styles)

Table 1: Requirements for a driver model

3.3 Research Setup

All driver models were developed using one reference vehicle and track. The reference trajectory was generated using the vPiL by applying synthetic accelerator and brake decelerations. This ensures that reference values for the actuators are available, and that the vehicle can attain the desired speed. The control theory-based driver models are parameterized using the reference track and reference car, and the RL-based driver model is trained using that same car and track.

To evaluate the generalizability of the two methods, two additional scenarios are accessible. To determine generalizability of the vehicle, a heavier and more powerful vehicle is driven on the given reference track (test 1). In a separate experiment, generalizability of the track is examined by driving the reference vehicle on a new track with additional driving maneuvers, such as speed steps (test 2).

To determine the control quality of the two driver models, the mean absolute error (MAE) is calculated. The control quality achieved by the control theory-based driver models, is used as a reference to evaluate the RL-based driver models.

4 Control Theory-based Driver Models

This chapter describes a control theory-based driver model. This driver model controls the current vehicle speed v_{act} to a reference vehicle speed v_{set} by setting accelerator t and brake b. In this chapter, two driver models are presented, which are increasingly complex to meet the requirements of chapter 3.2 with increasing quality. Both driver models consist of a finite state machine, which can assume the states acceleration and deceleration. In both states, a PI-controller controls the vehicle speed. In both driver models, the PI-controller input is the speed difference dv between the reference vehicle speed v_{set} and the current vehicle speed v_{act}:

$$dv = v_{set} - v_{act}$$

4.1 Baseline Model with Realistic Driver Behavior

The controller output in the acceleration state is an accelerator setpoint t, in the deceleration state it is a brake setpoint b, as shown in Figure 2. To reduce the number of state transitions, a speed tolerance v_{tol} is introduced. The state transition from acceleration to deceleration is activated when dv is less than v_{tol} and analogously, the state transition from deceleration to acceleration is triggered when dv is higher than v_{tol}:

ACCELERATION → DECELERATION: $[dv < v_{tol}]$
DECELERATION → ACCELERATION: $[dv > v_{tol}]$

To prevent integrator windup, the I-component of the PI-controller is limited as soon as the accelerator setpoint or brake setpoint have reached their control value limits.

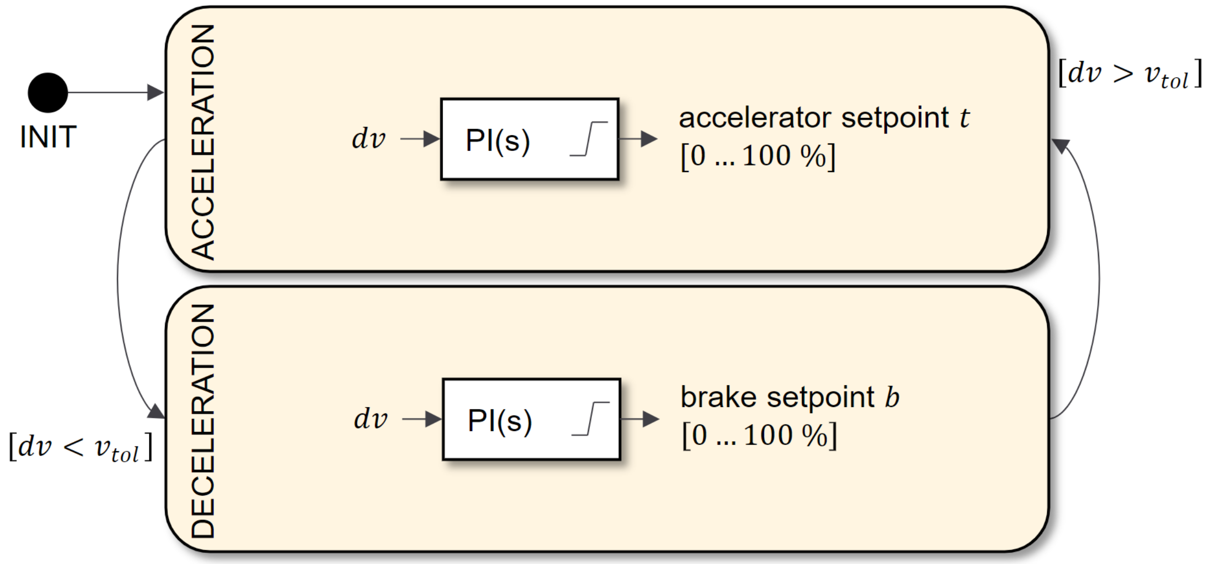


Figure 2: Baseline model using PI-controller with limited control values in a finite state machine.

Figure 3 shows the results of the baseline model. It is noticeable that the absolute deviations are always greatest when switching between ACCELERATION and DECELERATION. This is because the speed tolerance v_{tol} prohibits state transitions until it is exceeded. The smaller v_{tol} becomes, the smaller the deviations become, but for very small values high-frequent transitions of ACCELERATION and DECELERATION occur, which does not reflect human driving behavior.

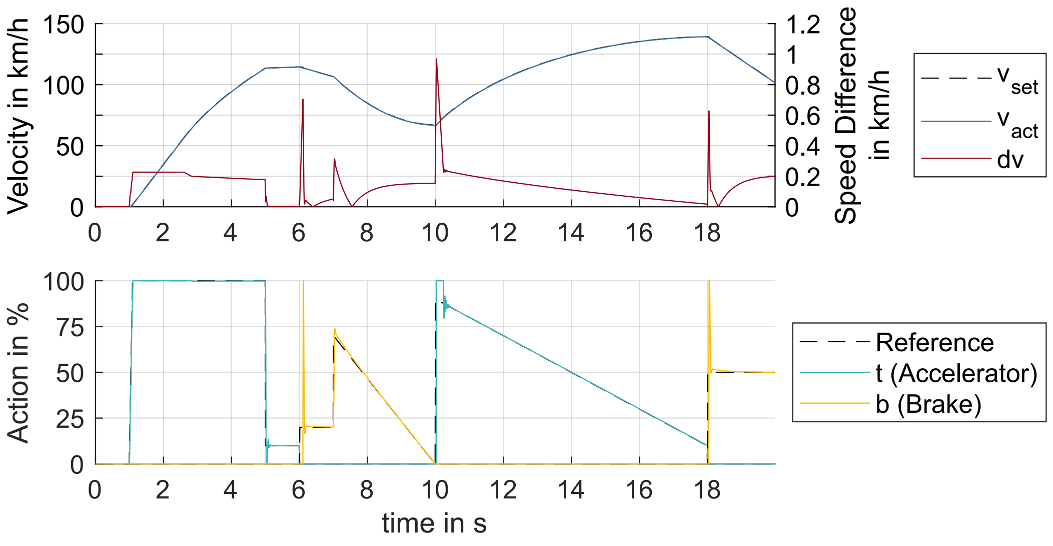


Figure 3: Velocity and Actions of the baseline model

4.2 Adaptive Driver Model using Pre-Control and Linearization

To improve the generalizability of the controller, the adaptive driver model uses pre-control and linearization of the controller output in the acceleration state, as shown in Figure 4. In this state, an engine torque T replaces the accelerator as the controller output. With the help of an inverted engine map, the appropriate accelerator setpoint t for the desired engine torque T can now be determined.

$$t = f(T)$$

This implementation creates a linear context between the controller input (differential speed dυ) and the controller output (engine torque T instead of accelerator setpoint) and thus improves the control behavior.

The pre-control calculates a reference acceleration a_{set} from the reference speed υ_{set}. This reference acceleration is converted into a reference force via the mass of the vehicle m and is added to the driving resistance forces $F_{roadload}$. The resulting force $F_{feedforward}$ is converted into a feedforward torque $T_{feedforward}$ for the acceleration state based on other vehicle parameters (dynamic wheel radius r and overall gear ratio i). For the deceleration state, the resulting force $F_{feedforward}$ is converted into a feedforward deceleration $a_{feedforward}$ based on the vehicle mass.

$$a_{set} = \dot{v}_{set}$$
$$F_{feedforward} = m \cdot a_{set} + F_{roadload}$$
$$T_{feedforward} = \frac{r}{i} \cdot F_{feedforward}$$
$$a_{feedforward} = -\frac{1}{m} \cdot F_{feedforward}$$

Since both model extensions include all relevant vehicle parameters (engine map, overall gear ratio, dynamic tire radius, vehicle mass and driving resistances), the generalizability of the driver model to other vehicles or tracks is increased.

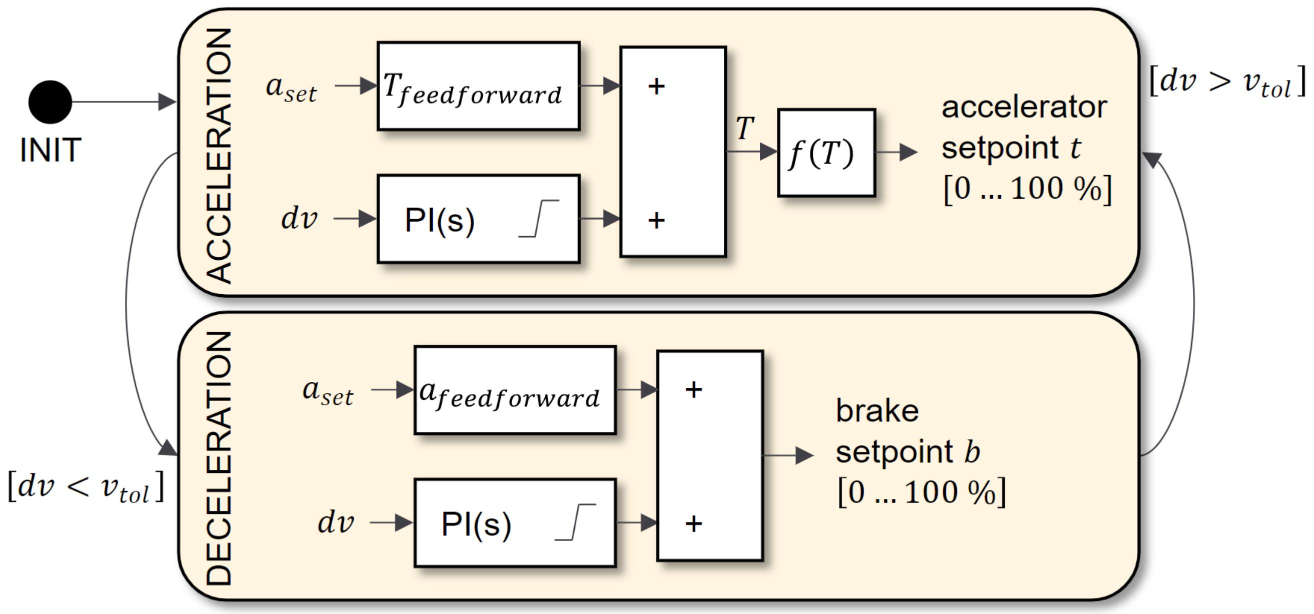


Figure 4: Adaptive driver model using feedforward control and linear controller outputs.

The adaptive driver model shows smaller deviations than the baseline model due to feedforward control and linear controller outputs, as shown in Figure 5. The absolute deviations are still greatest when switching between ACCELERATION and DECELERATION, but feedforward control minimizes the deviations caused by the state transitions. Note that the speed tolerance v_tol is set to the same value as in the baseline model.

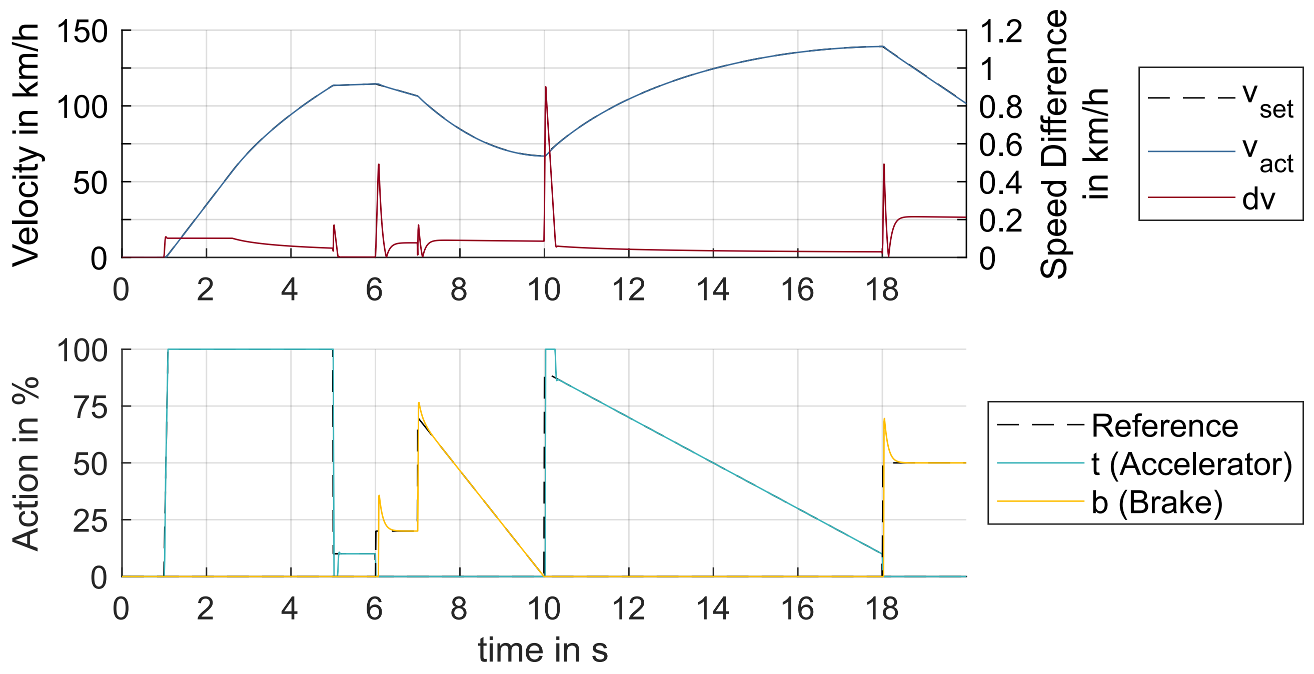


Figure 5: Velocity and Actions of the adaptive driver model

5 Reinforcement Learning-based Driver Model

In Reinforcement Learning (RL) an agent (≙ the driver) interacts with an environment (≙ the vehicle), as shown in Figure 6. By taking an action (≙ accelerating and decelerating), the agent changes its state (≙ current velocity) and receives a reward (≙ e.g. deviation from reference speed). If the agent receives a high reward, a similar action is done next: The action is reinforced. Otherwise, the agent tries a different action. In an iterative manner, the agent learns optimal behavior, an optimal policy, over time.

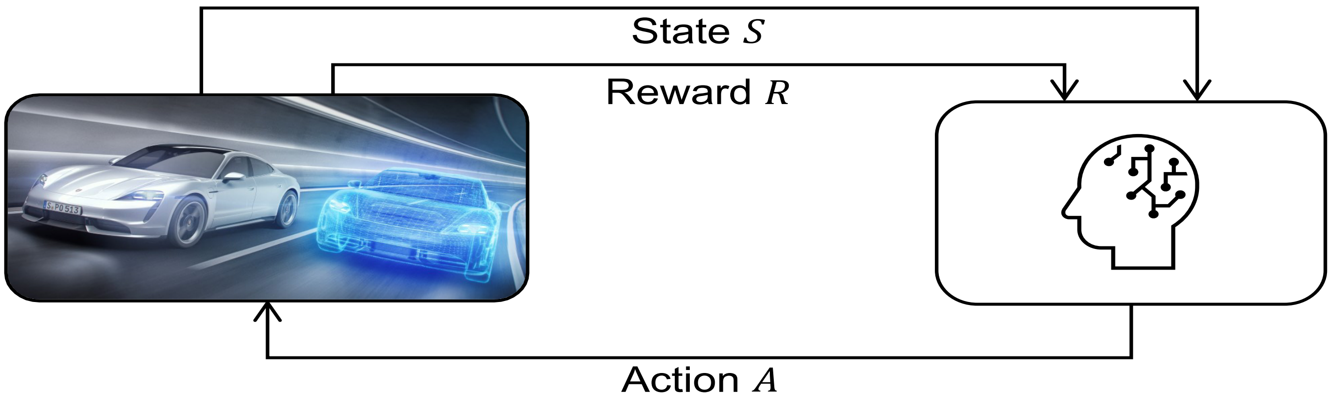


Figure 6: Reinforcement workflow of agent interacting with its environment.

A Reinforcement Learning agent is modeled as a Markov Decision Process (MDP) which is the tuple (S,A,R,p). Here, S ϵ $\mathbb{R}^d$ is the state space, A ϵ $\mathbb{R}^d$ is the action space, R: S×A→$\mathbb{R}$ is the reward function and p: S×A→S is the transition function of the environment. The RL-based driver model's quality can be modified by altering this tuple. More information regarding the individual elements can be found below.

The state space can consist of different time series containing either measured or calculated signals. Let t $\mathbb{N}$ be the time, $v_t^{act} \in \mathbb{R}$the actual speed of the vehicle, $v_t^{set} \in \mathbb{R}$ the reference speed. Due to the known trajectory, additional values $\{v_t^{set}, v_{t+1}^{set}, \dots, v_{t+N}^{set}\}$ are provided, where N ϵ $\mathbb{N}$ is a hyperparameter controlling the length of the foresight. The calculated state spaces used in this paper are the speed deviation $dv_{t+i}\ i \in \{1, \dots, M\} := v_t^{act} - v_{t+i}^{set} \in \mathbb{R}^M$ and the feedforward force $F_t^{feedforward}$ as calculated in chapter 3. For positive values only, the states are normalized between [0 1], all other states are normalized between [-1 1].

With t_t ϵ $\mathbb{R}$ and b_t ϵ $\mathbb{R}$ we denote the accelerator and vehicle deceleration outputs of the driver model and for the initial action space, we set:

$$A = \{t_t, b_t\} \in \mathbb{R}^2$$

In the following, we discuss how to ensure realistic driver behavior. To limit the driver model from accelerating and braking at the same time we change the action space to just one action:

$$A = action \in [-1,1]$$

Next, we reparametrize:

$$t_t = \mathbb{1}\{action \geq 0\} \cdot action$$
$$b_t = \mathbb{1}\{action < 0\} \cdot action$$

where the accelerator and brake values are now normalized in [0,1] and [-1,0), respectively. Even further we want to limit the variance of inputs. Therefore, we use generalized state-dependent exploration (gSDE) [10].

As reward function, we set:

$$r(s) = -|dv_t| \in \mathbb{R}$$

5.1 Evaluation of Different State Spaces

The driver model's actions are predefined by the described settings. The reward function already includes the most crucial requirement that the driver model must adhere to. As a result, we explore various combinations of input variables and the corresponding information required by the RL approach.

Figure 7 illustrates the driver model's performance under the initial setup, where we set the state space to:

$$S_1 = \{v_t^{set}, v_t^{act}\} \in \mathbb{R}^2$$

Therefore, gSDE has not been activated to illustrate the disparity in actions compared to when gSDE is activated. Table 2 displays the ratings of a driver model with an equivalent state space and activated gSDE.

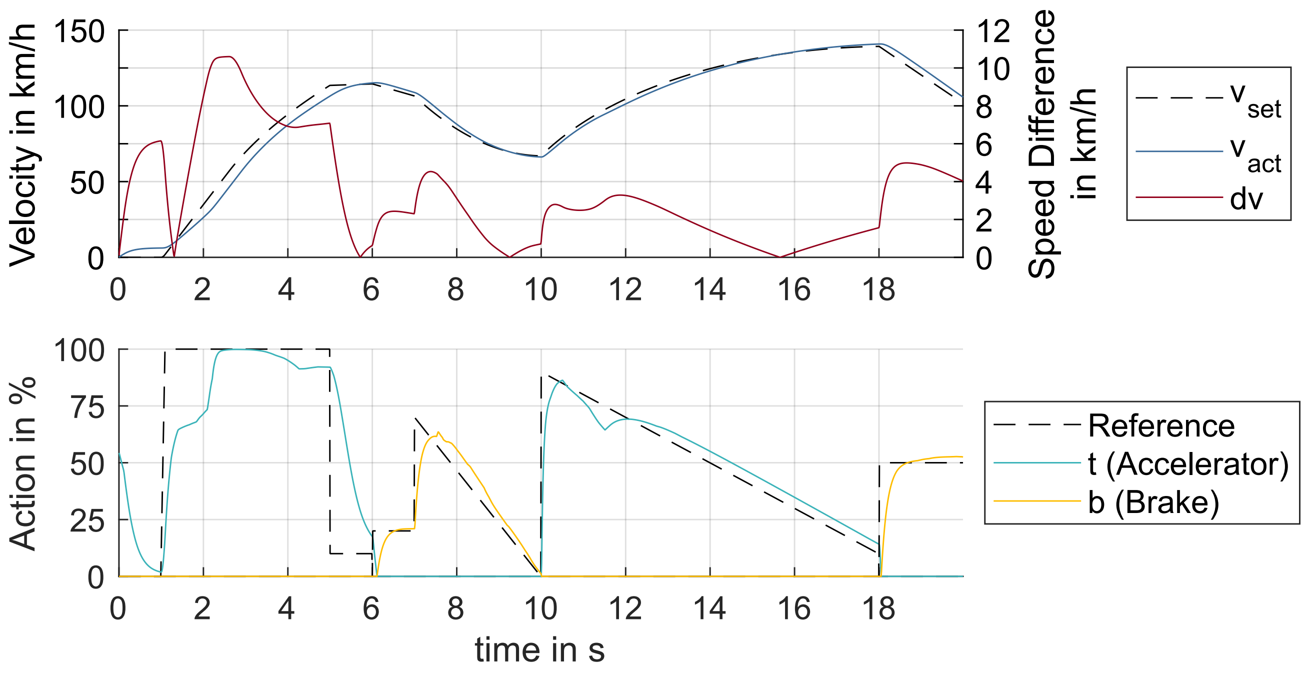


Figure 7: Velocity and Actions of the RL-based driver model with S_1

During the initial acceleration phase of the trajectory, there is a deviation of up to 10 km/h from the reference speed. The deviations result from the driver model applying the accelerator pedal during initial acceleration and only gradually reaching 100% acceleration. Even without gSDE activated, there is minimal variation in actions observed. However, it is noticeable that the MAE decreases when gSDE is activated.

Therefore, Figure 8 displays the outcomes of an analysis utilizing gSDE and setting the state space as:

$$S_2 = \{v_t^{set}, v_t^{act}, dv_t\} \in \mathbb{R}^3$$

Through dv, the agent obtains a superior input that links directly to the reward function.

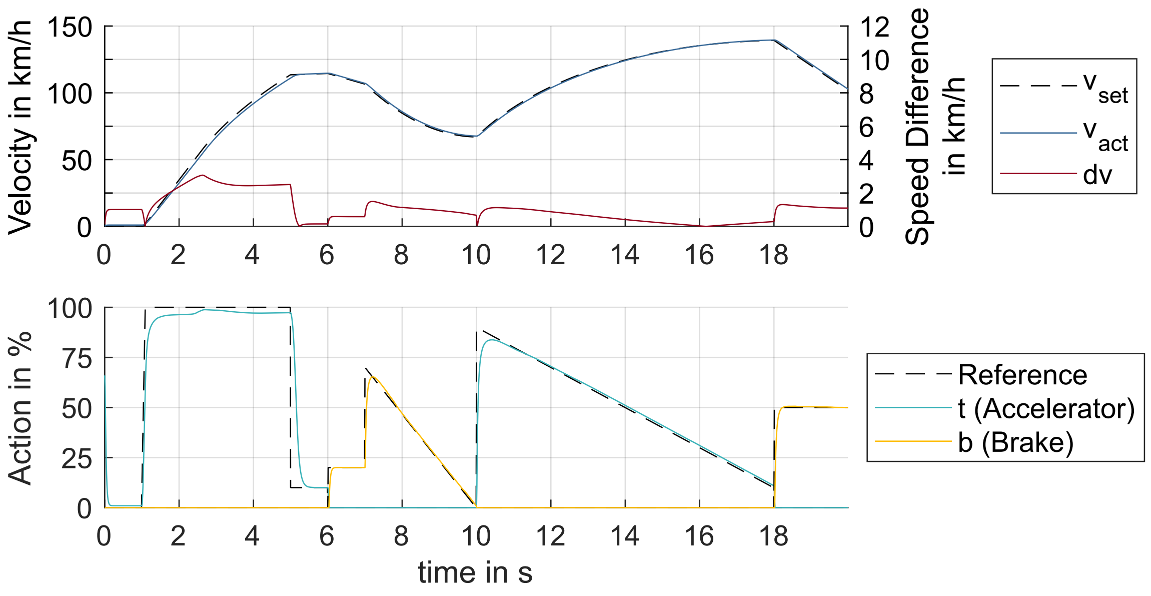


Figure 8: Velocity and Actions of the RL-based driver model with S_2

Due to the additional information, the deviations compared to the previous study are reduced by a factor of 3. While the gSDE enhances the results of the driver model, it also results in deviations at the acceleration and deceleration switching points. The hyperparameter generates smoother actions but impedes learning the required steps in the actions.

The upcoming study's state space is defined by

$$S_3 = \{v_t^{set}, v_t^{act}, dv_t, dv_{t+1}, \ldots, dv_{t+N}\} \in \mathbb{R}^{N+2}$$

where N=10. In comparison to control theory, Reinforcement Learning facilitates the integration of future value information into the driver model, which can be included in the state space.

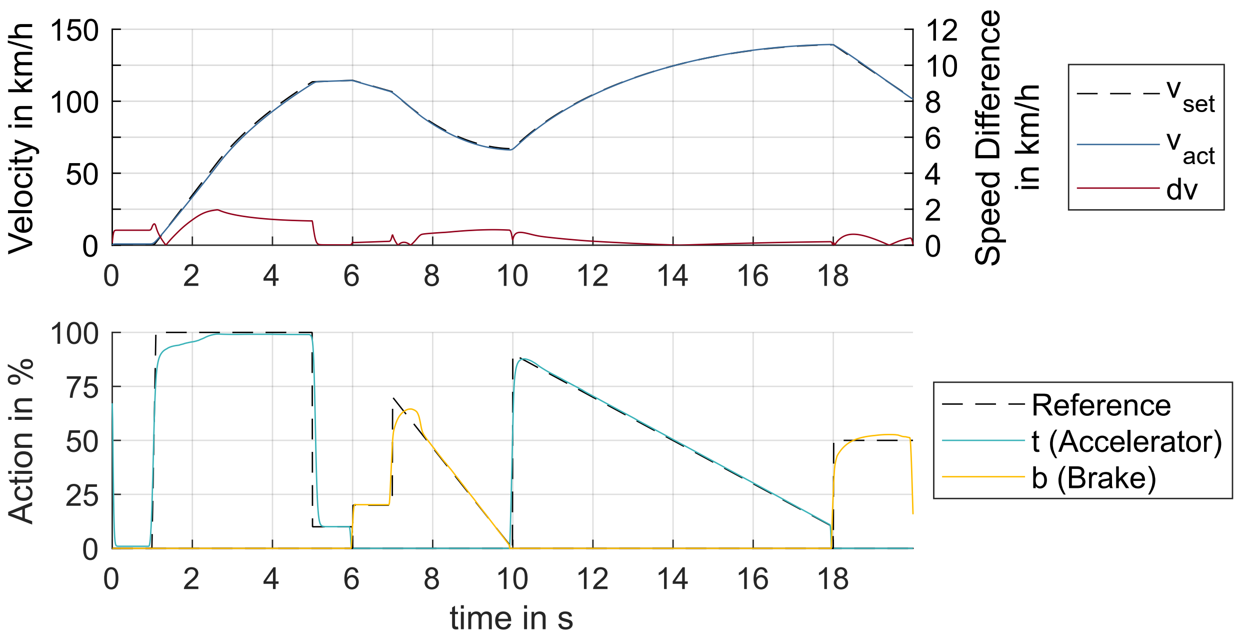


Figure 9: Velocity and Actions of the RL-based driver model with S_3

The mean absolute deviation can be reduced by 50 %. The future trajectory section provides crucial information that enables the driver model to learn and adopt a predictive behavior. To achieve this, the driver model initiates the transition between actions before a speed difference occurs.

Table 2 documents additional configurations of the state space. While implementing a feedforward control produces noteworthy results in the control approach, it does not yield any improvement in the RL-based driver model.

5.2 Training MathWorks® SIMULINK-based Agents in Python

Shown below is our high-level workflow to train an agent in Python using a SIMULINK-based environment. First, we convert the SIMULINK environment model into a Dynamic Linked Library (DLL). A DLL is a stand-alone collection of C functions and headers that we can directly execute in Python. See [11] for an example. Next, we build the environment in Python using the Gym library [12] and train the agent using Soft Actor-Critic [13] and Stable-Baselines3 [14]. Once the agent is trained, we deploy the learned policy in SIMULINK using ONNX [15].

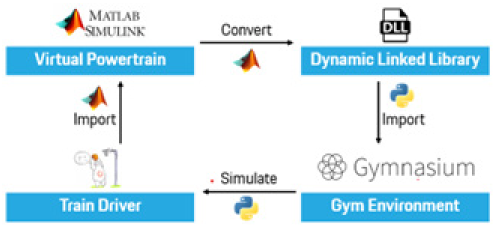


Figure 10: Setup for training MathWorks® SIMULINK-based Agents in Python

6 Comparison of Approaches

After analyzing the performance of two driver models on the training trajectory, this chapter compares the two approaches. To achieve this, Table 2 displays the results of both driver models, including the error rates of the control theory-based driver models as well as several RL-based driver models that use different state space combinations. The adaptive driver model exhibits the highest accuracy, achieving 0.04 km/h on the training trajectory. The performance of all models in Test 1 deviates by a maximum of 3% from the training results. The minor enhancements of some RL-based driver models are due to the increased performance of the test vehicle. Consequently, the driver model no longer needs to apply 100% accelerator pedal from second 1 to 5 to achieve the reference speed, allowing to counteract deviations.

Figure 11 presents the outcomes of the two driver models with the least deviations in test 2. The trajectory features characteristics that are absent from the training. In terms of speed, the driver model with adaptive control consistently shows the smallest deviations. Nevertheless, at state transitions (around 8 s and 12 s), abrupt increases in accelerator setpoints occur (up to 90 %), as the delayed transition due to v_tol necessitates compensation for the resulting deviation. The RL-based driver model performs effectively on the new trajectory with unknown characteristics, but in some situations (deceleration from 38 s) the deviations are larger than in training. To minimize these errors, retraining the driver model is an option. Retraining can be done on the new trajectory with the existing driver model and thus requires a shorter training time. In contrast to the control theory approach, the RL-based driver model can handle state transitions with smooth actions, because it includes a foresight function.

	model input configuration					mean absolut error [km/h]			number of action changes accelerator <> brake		
	M	N	v_act_t	F_tff	gsde	Training	Test 1	Test 2	Training	Test 1	Test 2
Reinforcement-Learning	10	1	yes	no	yes	0,53	0,53	18,40	3	3	8
	1	10	yes	no	no	0,83	0,82	18,73	4	4	12
	0	10	yes	no	yes	0,97	0,95	17,90	4	4	8
	1	1	yes	no	yes	1,03	1,02	18,09	3	3	8
	10	1	yes	no	no	1,22	1,22	44,81	4	4	9
	1	1	no	no	yes	1,22	1,21	18,12	4	4	8
	1	1	yes	yes	yes	1,34	1,33	19,93	6	6	8
	1	0	yes	no	yes	1,37	1,35	18,32	4	4	8
	1	1	yes	yes	no	1,61	1,59	20,81	6	6	8
	0	1	yes	no	yes	1,73	1,71	20,14	4	4	8
	0	1	yes	no	no	3,30	3,30	20,68	4	4	7
Control Theory	1	0	no	no	-	0,11	0,10	17,32	4	4	8
	1	0	no	yes	-	0,03	0,04	17,19	4	4	8

Table 2: Results of the two driver models

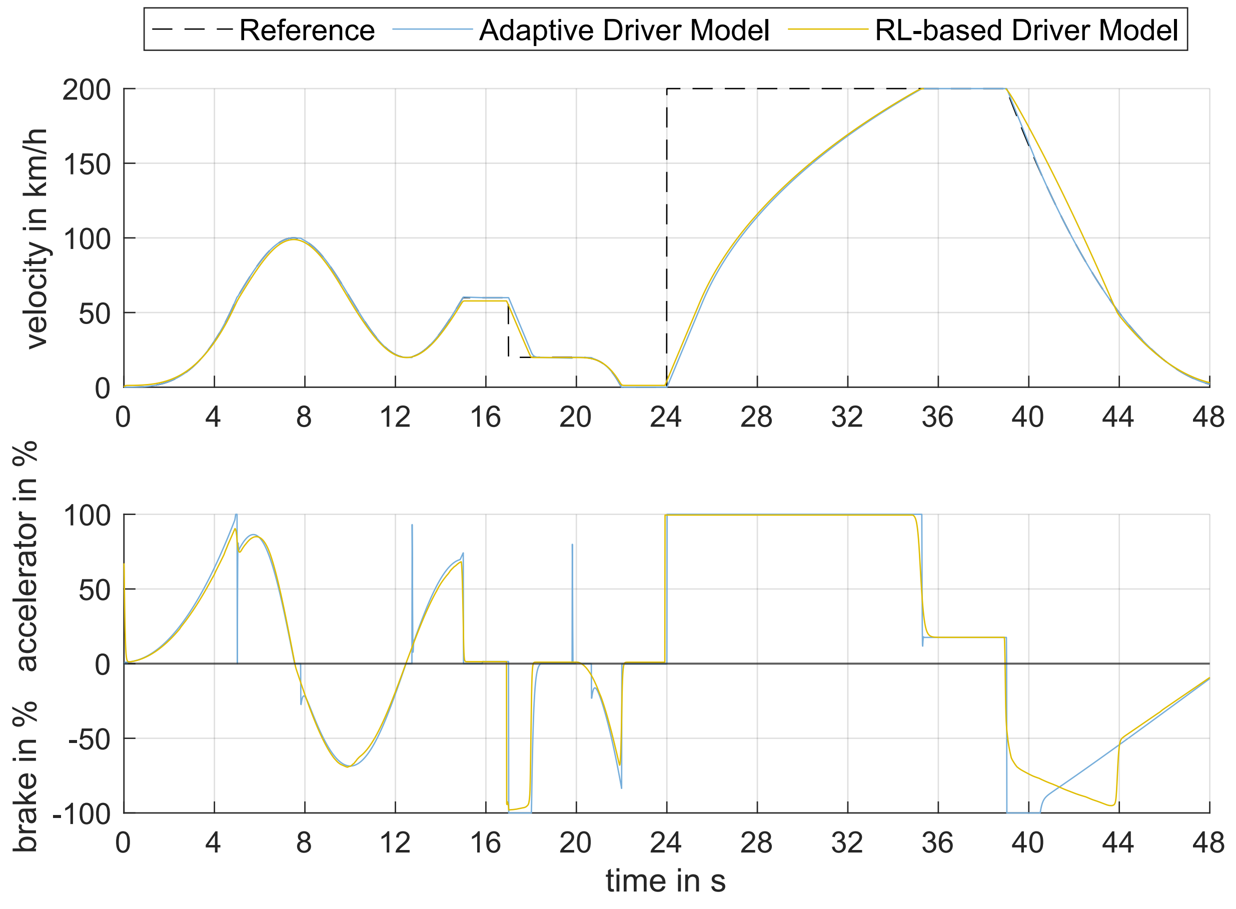


Figure 11: Velocity and Action of both driver Models in test 2

Figure 12 shows a qualitative comparison between the two driver models. The values range from 0, denoting low quality, to 5, indicating high quality. The control theory-based driver model showed slightly better control quality and is easier to interpret. Parameterization effort before use on the test bench and computational effort in use are lower due to the computationally intensive training of the RL-based driver model. Test 1 (modified vehicle parameters) and test 2 (modified reference vehicle speed trajectory) showed a comparable generalizability of both models. Due to the implemented foresight, the RL-based driver model was able to show a more realistic driving behavior.

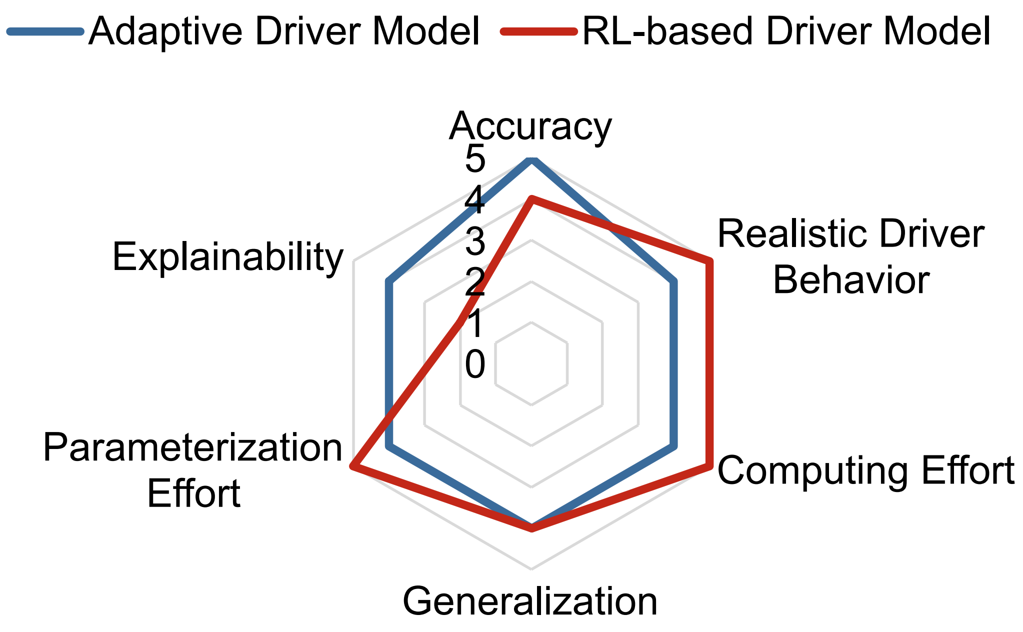
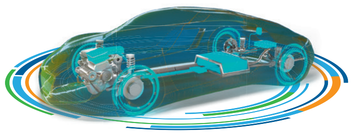


Figure 12: Evaluation of each best driver model.

For the application under consideration, the disadvantages of the RL-based driver model currently outweigh the advantages. Driver modelling at the 2nd level (see Chapter 2) also requires different driving styles (e.g., energy-saving, aggressive). In the case of an RL-based driver model, these could simply be achieved via the reward function (see Chapter 5), which would not be feasible in a control theory-based driver model.

7 Conclusion

A driver model based on reinforcement learning was developed within the vPiL, which controls a reference vehicle speed by setting accelerator and brake. The training of the RL-based driver model takes place virtually, the trained driver model in the form of a neural network is then integrated into the real-time environment of a PiL test bench. The driver model was compared with an existing control theory-based driver model and was able to achieve a slightly worse control quality. The disadvantage of computationally intensive training in advance is countered by more realistic driver behavior. It is also conceivable that the RL-based driver model could be used for driving tasks that cannot be described solely by specifying a target speed and curvature. Furthermore, a combination of both approaches could combine the respective advantages and should therefore be investigated.



Literature

[1] DONGES, Edmund: Fahrerverhaltensmodelle. In: WINNER, Hermann; HAKULI, Stephan; LOTZ, Felix; SINGER, Christina (Hrsg.): Handbuch Fahrerassistenzsysteme. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 17–26

[2] MICHON, John A.: A Critical View of Driver Behavior Models: What Do We Know, What Should We Do? In: EVANS, Leonard; SCHWING, Richard C. (Hrsg.): Human Behavior and Traffic Safety. Boston, MA: Springer US, 1985, S. 485–524

[3] WERLING, Moritz: Optimale aktive Fahreingriffe: De Gruyter, 2017

[4] KEBBATI, Yassine; AIT-OUFROUKH, Naima; VIGNERON, Vincent; ICHALAL, Dalil; GRUYER, Dominique: Optimized self-adaptive PID speed control for autonomous vehicles. In: YANG, Chenguang (Hrsg.): System intelligence through automation & computing: 2021 the 26th International Conference on Automation & Computing: University of Portsmouth, Portsmouth, UK, 2nd-4th September 2021. Piscataway, NJ: IEEE, 2021, S. 1–6

[5] AZZAGHDAM, Elif Toy; ALANKUS, Orhan Behic: Longitudinal Control of Autonomous Vehicles Consisting Power-Train With Non-Linear Characteristics. In: IEEE Transactions on Intelligent Vehicles 7 (2022), Nr. 1, S. 133–142

[6] MAYR, Christian; MERL, Reinhard; GIGERL, Hans-Peter; TEITZER, Mario; KÖNIG, David; STEMMER, Daniel; RETTER, Felix: Test emissionsrelevanter Fahrzyklen auf dem Motorprüfstand. In: LIEBL, Johannes (Hrsg.): Simulation und Test 2018. Wiesbaden: Springer Fachmedien Wiesbaden, 2019 (Proceedings), S. 107–125

[7] BUECHEL, Martin; KNOLL, Alois: Deep Reinforcement Learning for Predictive Longitudinal Control of Automated Vehicles. In: 2018 IEEE Intelligent Transportation Systems Conference: November 4-7, Maui, Hawaii. Piscataway, NJ: IEEE, 2018, S. 2391–2397

[8] WANG, Bin; ZHAO, Dongbin; LI, Chengdong; DAI, Yujie: Design and implementation of an adaptive cruise control system based on supervised actor-critic learning. In: 2015 5th International Conference on Information Science and Technology (ICIST) : 24 - 26 April 2015, Changsha, China. Piscataway, NJ : IEEE, 2015, S. 243–248

[9] VEITH, Jan-Michael; SCHILLING, Jannes: Parametrierung und Optimierung eines Fahrreglers mittels virtuellem Antriebs-strangprüfstand. In: 9. AutoTest Fachkonferenz.

[10] RAFFIN, Antonin; KOBER, Jens; STULP, Freek: Smooth Exploration for Robotic Reinforcement Learning. In: Proceedings of the 5th Conference on Robot Learning. URL http://arxiv.org/pdf/2005.05719v2

[11] DAPPERFU: Python-Simulink. URL https://github.com/dapperfu/Python-Simulink – Überprüfungsdatum 2023-09-17

[12] BROCKMAN, Greg; CHEUNG, Vicki; PETTERSSON, Ludwig; SCHNEIDER, Jonas; SCHULMAN, John; TANG, Jie; ZAREMBA, Wojciech: OpenAI Gym. 05.06.2016

[13] HAARNOJA, Tuomas; ZHOU, Aurick; ABBEEL, Pieter; LEVINE, Sergey: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. 04.01.2018

[14] RAFFIN, Antonin; HILL, Ashley; GLEAVE, Adam; KANERVISTO, Anssi; ERNESTUS, Maximilian; DORMANN, Noah: Stable-Baselines3: Reliable Reinforcement Learning Implementations. In: Journal of Machine Learning Research 22 (2021), Nr. 268, S. 1–8. URL http://jmlr.org/papers/v22/20-1364.html

[15] BAI, Junjie; LU, Fang; ZHANG, Ke; OTHERS: ONNX: Open Neural Network Exchange. URL https://github.com/onnx/onnx