Received September 29, 2020, accepted October 11, 2020, date of publication October 14, 2020, date of current version October 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031037

Hierarchical Evasive Path Planning Using Reinforcement Learning and Model Predictive Control

ÁRPÁD FEHÉR, SZILÁRD ARADI, (Member, IEEE), AND TAMÁS BÉCSI, (Member, IEEE)
Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, 1111 Budapest, Hungary

Corresponding author: Tamás Bécsi (becsi.tamas@mail.bme.hu)

This work was supported in part by the National Research, Development, and Innovation Office (NKFIH) through the Project ''National Laboratory for Autonomous Systems'' under Grant NKFIH-869/2020, and in part by the Hungarian Government and co-financed by the European Social Fund through the Project ''Talent Management in Autonomous Vehicle Control Technologies'' under Grant EFOP-3.6.3-VEKOP-16-2017-00001.

ABSTRACT Motion planning plays an essential role in designing self-driving functions for connected and autonomous vehicles. The methods need to provide a feasible trajectory for the vehicle to follow, fulfilling different requirements, such as safety, efficiency, and passenger comfort. In this area, algorithms must also meet strict real-time expectations, since, especially in an emergency, the decision time is limited, which raises a trade-off for the feasibility requirements. This article proposes a hierarchical path planning solution for evasive maneuvering, where a Twin Delayed DDPG reinforcement learning agent generates the parameters of a geometric path consisting of chlotoids and straight sections, and an underlying model predictive control loop fulfills the trajectory following tasks. The method is applied to the automotive double lane-change test, a common emergency situation, comparing its results with human drivers' performance using a dynamic simulation environment. Besides the test's standardized parameters, a broader range of topological layouts is chosen, both for the training and performance evaluation. The results show that the proposed method highly outperforms human drivers, especially in challenging situations, while meeting the computational requirements, as the pre-trained neural network and path generation algorithm can provide a solution in an instant, based on the experience gained during the training process.

INDEX TERMS Vehicle dynamics, advanced driver assistance systems, machine learning, artificial neural networks, reinforcement learning.

I. INTRODUCTION

Trajectory and path planning methods play a significant role in designing different autonomous vehicle functions. In the last decades, industrial and academic participants made substantial efforts in developing different highly automated functions as part of Advanced Driver Assistance Systems (ADAS) as well as in the development of Connected and Automated Vehicles (CAVs). As these systems aim to carry passengers, they have to fulfill different requirements, from which the most important is safety, though they must ensure passenger comfort and customization capabilities also. As part of the safety goals, such systems need to solve hazardous situations in avoiding potential accidents, or at least minimize its casualties. Moreover, under dangerous situations, the autonomous algorithms have to provide a feasible maneuver, i.e., trajectory or path, in an instant, which raises the need for real-time methods. Though it is not possible to define all possible hazardous situations, standardization organizations have a collection of previously described test cases to validate the vehicles' and algorithms' performance, which are tested on automotive proving grounds [1]. One is the double lane-change test (DLC) defined in ISO-3888-2 [2]. Originally known as the ''Moose test'', it aims to determine how well a specific vehicle evades a suddenly appearing obstacle.

The ISO double lane-change test formulates as follows. It starts with an entry lane with a length of 12m and ends with an exit lane with the same size. Between these two, there is a so-called side lane, with a lateral offset of 1m. The longitudinal distance between the entry and side lanes is 13.5m, while between the side and exit lanes is 12.5m. The test also defines that 2 meters after entering the scene, the throttle has

The associate editor coordinating the review of this manuscript and approving it for publication was Jianyong Yao.

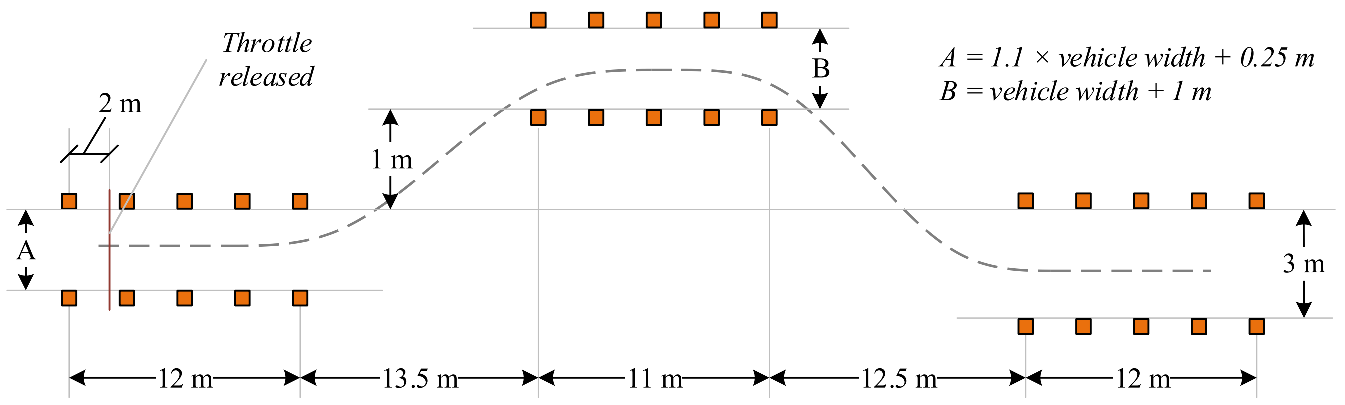


FIGURE 1. The double lane-change maneuver according to ISO-3888-2 [2].

to be released, and the rest of the test is carried out without any throttle or brake actuation, solely with steering. The tests are usually driven with and without Electronic Stability Program (ESP). Fig. 1 shows the geometric layout of the test.

This article presents a solution based on Reinforcement Learning for a point-free avoidance path generation for the emergency double-lane change situation. A model predictive control (MPC) based machine driver steers the underlying dynamic vehicle model. The generated algorithm results are compared to the performance of human drivers carried out in a simulator environment.

The commonly used hierarchical structure of autonomous driving starts with route planning. It follows with perception and localization aided by Vehicle-to-X (V2X) communication if present. This defines the vehicle's state and the position and movement information on other traffic participants, such as traffic topology, surrounding vehicles, pedestrians, and other dynamic or static objects or obstacles. These add together to a scene representation in which the automated system needs to make decisions. In an underlying layer, behavior and scene prediction could further help the process. High-level actions are chosen on the strategic level, such as lane-changing, car-following, avoidance maneuver, etc. This provides the trajectory planner's input, which has to generate a feasible path for a short time interval. On the lowest level, a trajectory following control loop provides the actuator inputs, i.e., steering, throttle, and brake commands. For an overview of the different levels of motion planning, see [3], [4].

A. RELATED WORK

Path planning generates a curve or a set of points for a vehicle to follow, while trajectory planning gives additional speed information along the path. These methods can be categorized into four main classes: geometry-based methods, heuristic solutions, and algorithms based on optimal control or nonlinear programming. Finally, in recent years, machine learning based solutions also emerge. Geometric methods use curve fitting based on the given start and end states and avoidable obstacles as inputs and generate the path as a combination of straight lines, circular arcs, or splines [5]. Though such methods are computationally fast, the generated path's dynamical feasibility is not guaranteed; hence, it has to be evaluated afterwards [6].

Several artificial intelligence based methods can be found among the heuristics-based approaches that use search or random sampling algorithms. Contrary to the geometric methods, these can be computationally expensive. For search-based methods, different extensions of the A* algorithm are applied. To reduce complexity, The authors of [7] used sample time based refining, while in [8], the authors use adaptive refining for the same purpose. As A* provides a discrete solution, which is not directly applicable for path planning of nonholonomic vehicle dynamics, several extensions, such as the Hybrid A* algorithm is used to associate a continuous state to the discrete representation, and hence, generating a path, that is easier to follow [9]. Random sampling methods on the field are usually based on the Rapidly-exploring Random Tree (RRT) algorithm, which is a generic heuristics for searching non-convex spaces with a tree structure, guided by random samples. Such methods take the vehicle's initial state and build a tree from feasible branches based on the dynamics constraints. As the tree reaches the desired end state, the corresponding set of branches define the path [10], [11].

To ensure both dynamical feasibility nonlinear optimization based methods define the problem as a nonlinear optimization problem (NLP), where the technique generates a path, generally based on some geometric method, and establishes a value function, how well the vehicle can travel along the path [12], [13]. This generates a constrained minimization problem that can be solved through different techniques. Though dynamical feasibility always raises the trade-off for computational complexity [14]. Using linear parameter varying (LPV) models and modern model predictive control (MPC) techniques can provide near-optimal solutions, though this trade-off still remains in this application domain [15].

Machine learning based solutions can mitigate this trade-off by training an agent to generate feasible paths.

One possible solution is the utilization of supervised learning. The mentioned NLP solver can generate many feasible paths for different constraints and goals and use these as a training dataset of a neural network, which can generalize the problem and create trajectories in real-time. Such a solution can be found in [16]. However, it is not always possible to generate datasets that are large enough for training. To solve this problem, reinforcement learning-based techniques use self-play with trial-and-error and trains the agent based on the experiments gained from performing a large amount of trials [17]. One subset of RL is the behavior cloning based methods, or inverse RL, where the agents try to generalize from demonstrations by learning a mapping between observations and actions. An overview of methods based on imitation learning for vehicle motion planning can be found in [18]. Reinforcement learning (RL) based methods usually use an end-to-end approach, trying to generate the direct steering, throttle, and brake commands based on the environmental information available. These researches use a various set of sensor models, such as grid-based topological [19], lidar-like beam sensors [20], camera information [21] or the high-level ground truth position information [22]. Another group of researches focus on strategic decisions, where the agent determines high-level actions, such as lane-change, follow, etc. These researches usually use microscopic simulations for the environment, such as Vissim [23], Udacity, [24], SUMO [25], or several self-made models [26]. Though hybrid solutions exist, where the strategic and direct control meets [27], only a few papers deal with defining a path by some geometric approach an RL and then drive through it with a controller [28], [29]. For further information on the topic, a good review of the RL based approaches for vehicle motion planning can be found in [30].

In recent years, many research focuses directly on the double lane-change problem. The DLC problem is solved as an optimal control problem in [31] considering a low degree-of-freedom vehicle model. The authors of [32] formulates the problem as a series of convex approximation, and an optimal trajectory that minimizes yaw acceleration is calculated and drives through it with an MPC controller. In [33], the authors use black-box modeling of double lane-change maneuver based on Adaptive Neuro-Fuzzy System (ANFIS) and presents a simplified test with an actual vehicle at low speed (30 km/h). In [34], the authors present an explicit MPC controller to reduce the enormous computational complexity of MPC by using an mp-QP technique and validates it in the CarSim simulator. To perform this, they assume an LTI vehicle model for the planning phase. Based on the investigation of real drivers' recorded maneuvers, the authors of [35] propose a hyperbolic tangent lane-change trajectory model and verified it using both real data and simulation. A game-theoretic approach is presented in [36], for a stochastic game-based steering torque control framework for a driver–machine copilot system described by an affine-LQ method based on a 6-order stochastic driver–vehicle dynamic system. Also, the Stackelberg-game-based shared control scheme for a path-tracking is presented in [37]. For path tracking, [38] propose a H_∞ controller considering look-ahead information, and validate it in CarSim. An indirect shared control under double loop framework is proposed in [39], dividing the driver and controller into two independent closed loops to obtain the weighted steering angle through an authority allocation system.

B. CONTRIBUTIONS OF THE PAPER

The paper presents a geometric emergency maneuver path planning agent for the double lane-change problem, based on a Deep Deterministic Policy Gradient (DDPG) variant, namely Twin Delayed DDPG (TD3). The agent defines a path from combined clothoid and straight sections using the TD3 algorithm. The path is evaluated by driving along with an MPC controller. The results are compared to the performance of human drivers.

Section II formulates the problem, and gives topological information on the task. The RL agent and the design structure is introduced in Section III-A, including the algorithm used, the network architecture, the state and action spaces, and the reward function used. In Section III-B, the used dynamic vehicle model is briefly presented, afterwards, Section III-C presents the path generation process, and Section III-D provides information on the path following MPC controller. Finally, Section IV provides information on the test environment and the results of the training and compares the agent's and human driver's performance in simulation on different tracks.

II. PROBLEM DESCRIPTION

The study presented in this article aims to give a possible implementation of a real-time optimal path planning of an emergency double lane-change maneuver. At the beginning of the research, the following simplifications have been introduced to focus on the primary problem.

- The maneuver is performed on a straight section of the road.
- The vehicle has ideal high-level sensor signals.
- Constant asphalt-wheel friction coefficient.

The goal is to plan an optimal path, from the initial state to the desired end state, taking the given environmental, physical constraints into account.

As mentioned before, The DLC test defines three lanes that must be passed by the vehicle, as shown in Fig. 2. The optimization condition is to maximize travel comfort by minimizing lateral acceleration and jerk. The speed of the vehicle in the initial state v_0, the lane widths w_1, w_2, w_3, the lengths l_1, l_2, l_3 and the positions (x_2, y_2), (x_3, y_3) of the green lanes can vary within the limits of vehicle performance. The vehicle's center of gravity (CoG) and the coordinate system's origin is placed in the middle of the entry lane. In addition to route planning, the goal is to provide a good estimate of feasibility by finding the optimum.

Our previous studies have already dealt with the topic of motion planning, as a spline-based route planner with

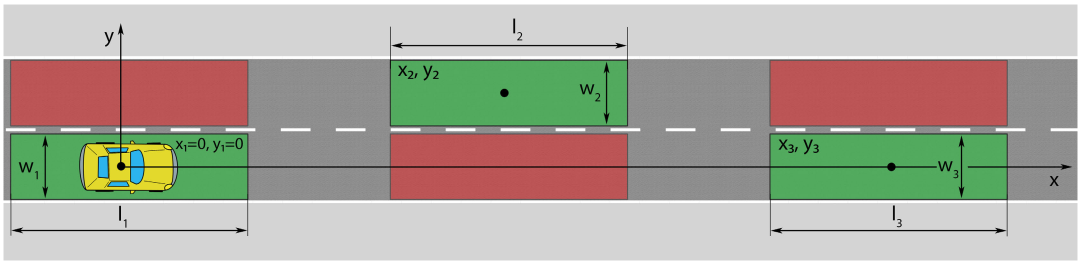


FIGURE 2. The topological layout of the double lane-change problem, with the varying parameters used for the training.

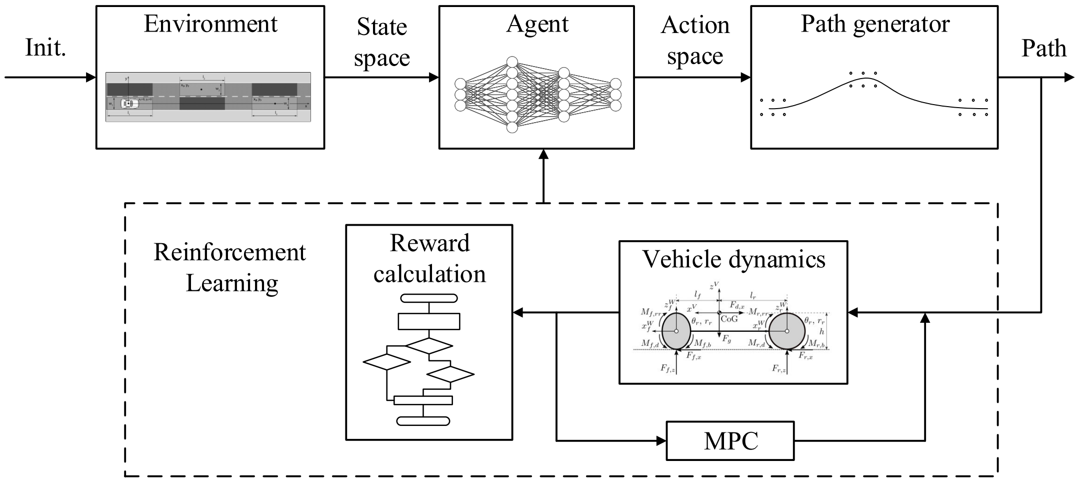


FIGURE 3. The training and control loop used for the design of the agent.

reinforcement learning [40] having tested on an actual vehicle [29]. Fig. 3 provides an overview of the agent used in this article. The research neglects the sensing part of the problem and starts with the extracted physical constraints as an input state space for the actor-network of the algorithm. The outputs of the network are the physical parameters that can be used for generating a path, described in Section III-C. The MPC control loop drives the vehicle among the given path, which is evaluated afterward to calculate the reward based on the success/failure information and additional performance parameters. This reward function closes the RL loop and is used to fit the agent's neural networks.

III. METHODOLOGY

This section details the methods used to solve the problem. For most machine learning algorithms, reinforcement learning also requires separate training and the evaluation phase. The methodology chapter focuses on the training of the neural networks.

A. REINFORCEMENT LEARNING ENVIRONMENT

In the traditional sense, the reinforcement learning system consists of an environmental model in which a learning agent must choose from the action-space correctly. Based on the environment's state-space, which serves as an input for the agent, it performs actions to receive a reward value based on its performance, called the reward function. During the training phase, the goal is to maximize future reward value. Visualization and rendering help to evaluate the results, enabling the human supervisor to determine the aspects in which the learned system can be further developed.

Most of the influential frameworks to build machine learning-based solutions are currently based on the Python programming language. Several state-of-the-art reinforcement learning agents (e.g., Intel Nervana Coach [41]), open-source machine learning platforms (e.g., Tensorflow, Keras), and test environments (e.g., OpenAI Gym) are available, which will make the development process more efficient. A reinforcement learning environment should be built to solve the problem outlined in Section II. The environment is written in Python programming language, and the constituent components are presented in the following subsections. All parts of the environmental model were implemented at the source code level.

1) DEEP DETERMINISTIC POLICY GRADIENT AGENT

The Deep Deterministic Policy Gradient Agent (Fig. 4) is a powerful model-free Reinforcement Learning algorithm with continuous action and state-space, based on the Deterministic Policy Gradient (DPG) algorithm [42]. It consists of an actor

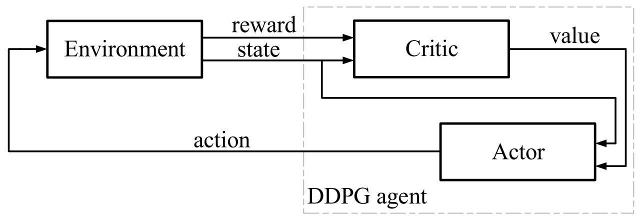


FIGURE 4. **Deep Deterministic Policy Gradient.**

and a critic network. The actor-critic combines value-based methods such as Deep Q-network (DQN) and policy-based approaches.

In the DDPG algorithm [43], [44], the actor and critic work with two different neural networks.

$$a = \mu(s; \theta^{\mu}) \tag{1}$$

In (1), the actor-network output is the action a, s stands for the state-space, and θ^{μ} are learning weights of the network. Using this, the actor gives a deterministic approximate of the optimal policy. The actor-network does not learn a probability distribution over actions, but the best-believed action in all states. The actor is learned the $argmax(Q(s, a))$ function, which is given the best action.

$$Q(s; a; \theta^{Q}) \Rightarrow Q(s; \mu(s; \theta^{\mu}), \theta^{Q}) \tag{2}$$

The critic-network (2) returns the Q value, where θ^{Q} stands for the weights of the network. The critic-network works with the output of the actor-network (the actor's best-believed action) and learns to evaluate it. The $\mu(s; \theta^{\mu'})$ and $Q(s; a; \theta^{Q'})$ are the target actor and critic networks, where $\theta^{Q'}$ and $\theta^{\mu'}$ are the weights. The actor-network is updated with policy gradients, while the critic-network is updated with gradients calculated from the Temporal Difference (TD) error.

$$\mu(s; \theta^{\mu}) + N \tag{3}$$

Noise N applied to the output of the actor provides the exploration. The Ornstein–Uhlenbeck process [45] is the most common method for noise production to DDPG.

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{4}$$

During training, the replay buffer is filled with the parameters of each step. The target Q value is updated with (4) by random sampling.

$$L = \frac{1}{M} \sum_i (y_i - Q(s_i, a_i|\theta^{Q})^2) \tag{5}$$

The TD error can be calculated by the (5) formula, which can be used to update the weights of critic-networks with gradients calculated from the L loss. The policy network is updated with the policy gradient method. DDPG uses a soft update strategy for greater stability, which is a slow blending between the regular and target network weights.

A one-step reinforcement learning was used to determine the parameters of the emergency evasive maneuver. This means that an episode consists of one step, and neural networks are updated after each step. This solution reduces the complexity of the task and allows the use of a large continuous state-space and many actions.

TABLE 1. **Hyperparameters of the trainig.**

Actor network	
Learning rate (α)	0.0001
Batch size	64
Hidden F.C. layer structure	[128,100,64]
Critic network	
Learning rate (α)	0.001
Discount factor (γ)	0.99
Head 1 hidden F.C. layer structure	[128, 64, 128]
Head 2 hidden F.C. layer structure	[128]
Ornstein-Uhlenbeck parameters	
(μ)	zeros(8)
(σ)	0.5
(θ)	0.5

2) NETWORK ARCHITECTURE AND HYPERPARAMETERS

The actor and the critic are two separate neural networks. The actor-network task is to provide the most optimal parameters for a given state-space to generate the trajectory. The critic neural network learns to evaluate the actions of the actor for improvements in the learning process.

The first element of the actor-network structure is an 11-element input layer. Three fully connected hidden layers follow this. For hidden layers, batch normalization and rectified linear unit (ReLu) activation function is used. The 8-element output layer has a hyperbolic tangent activation function. Table 1 summarizes the architecture of both networks. The structure of the critic-network is more complex, having input layers. An 11-element input represents the states, and an 8-element describes the actions defined by the actor. Three hidden layers follow the 11-element input layer with batch normalization and a ReLu activation function. A fully connected hidden layer follows the 8-element layer. The topology of the last two layers of both networks are the same, and their weights and biases are summarized. Finally, a single output ReLu activation layer closes the network.

3) STATE-SPACE

As the training paradigm is model-free, the agent has no information about the environmental model, only receives quantified state variables closely related to the actions. To solve this article's basic problem, the status variables are to specify the vehicle's initial speed and designate an area over which the vehicle must travel most optimally. The state-space (6) consists of 11 continuous states.

$$[v_0, l_1, w_1, x_2, y_2, l_2, w_2, x_3, y_3, l_3, w_3] \tag{6}$$

The first is the vehicle speed v_0 at the torque release line. Three lanes are defined (see Fig. 2), for which w_i and l_i are the width and length of the lanes to stay within, and x_i, y_i are the points of the lane center in the coordinate system in which the initial lane position is the origin. Based on the experience gained with previous reinforcement learning systems, it is worth normalizing the states to the range of [0, 1]. If one of the state variables takes orders of magnitude higher or lower, the neural network training may be more time consuming or even learn in the wrong direction. Normalization is performed with a min-max scaler.

4) ACTION-SPACE

The agent has eight continuous action outputs, which significantly increases the complexity of the training task. The action parameters determine the route. The route of the overtaking maneuver is always composed of pre-defined segments III-C. These sections have longitudinal and lateral distances that can be fully defined, but the agent's action-space does not directly specify these distances. During the development, experience shows that training gives much sooner and significantly better results if the action space [0,1] is normalized. Fig. 7 shows the length of the first straight section s_1, the lateral and longitudinal distances of the first curved section x_{c_1}, y_{c_1}, the length of the second straight section s_2, the lateral and longitudinal distances of the second straight section x_{c_2}, y_{c_2}, and the length of the third straight section s_3 must be specified in order to generate the route. These distances were determined by ratio values for normalization with knowledge of the path's endpoint, which is defined to the center of the last lane.

Ratio parameters are added to the action-space (7) to determine the longer distances of the track. The s_2, y_{c_1}, y_{c_2} are short distances that can be obtained by multiplying the normalized action by a small number. An additional benefit of introducing ratio variables is reducing the number of dynamically infeasible paths during the training phase.

$$[s_2, r_{sc_{11}sc_{32}}, r_{s_1c_1}, r_{s_3c_2}, y_{c_1}, y_{c_2}, p_1, p_2] \tag{7}$$

The (8) can be used to determine the distances required to generate the path from the ratio variables.

$$\begin{aligned} x_{s_1c_1} &= (x_3 + \frac{l_3}{2} - l_2)r_{sc_{11}sc_{32}}, \\ x_{s_3c_2} &= (x_3 - x_{s_1c_1} - l_2), \\ x_{s_1} &= x_{s_1c_1}r_{s_1c_1}, \quad x_{c_1} = x_{s_1c_1} - x_{s_1}, \\ x_{s_3} &= x_{s_3c_2}r_{s_3c_2}, \quad x_{c_2} = x_{s_3c_2} - x_{s_3} \end{aligned} \tag{8}$$

5) REWARD FUNCTION

The learning process consists of a series of iterations. It consists of a series of steps in which the success of the attempts can be characterized by reward value. In an emergency overtaking maneuver, the reward function's task is to quantify the goodness of a route, in which the agent responds to a given state. The reward function is built according to the optimization goals.

The reward procedure is unusual and complex for this problem. A classical controller III-D and a near-realistic, efficient runtime vehicle model III-B is implemented as part of the rewarding process. For each evaluation step, the vehicle model is driven along the planned track in a classical control loop with an MPC controller. The reward value is determined by the control errors and the vehicle model's dynamic parameters as it is driven along the track.

Negative rewards are given to the agent in the following termination cases:

- Longitudinal rear or front slip greater than 0.2
- Lateral rear or front slip greater than 0.15
- The euclidean distance between vehicle COG and the nearest point of the path (distance error) greater than 3 meter
- The angle difference between at closest point of the path and vehicle (angle error) greater than 40 degree
- The vehicle does not cross the green lane (Fig. 2) or leaves it sideways

The value of the negative reward is -1.5 in all cases. If the vehicle reaches the end of the planned course, the agent gets a positive reward (10). The optimization condition is to minimize lateral acceleration. The lateral slip is proportional to the lateral acceleration, so small front and rear lateral slip values μ_{lf}, μ_{lr} are rewarded. The value of the maximum allowable slip depends mostly on the speed. An empirical formula is defined to make the reward value speed-independent over the entire range. A formula (9) gives a maximum slip value depending on the vehicle's initial speed. The agent achieves greater efficiency in higher speed cases with this solution.

$$\mu_{max} = 0.0037e^{v_0\, 0.0693} \tag{9}$$

The reward function (10) subtracts the maximum values that occur during the route's execution from the experience maximum value.

$$r = (2\mu_{max}) - \mu_{lf} - \mu_{lr} \tag{10}$$

6) RENDERING AND CARLA INTEGRATION

Since the development of automotive functions raises several safety issues, their testing usually consists of a simulation phase, where the potential conflicts can be evaluated in a safe environment [46]. During the development of a learning agent and environment, using theoretical knowledge, the experience should be gained about the agent's behavior in the created environmental model, based on which the reward procedure should be refined and modified the hyperparameters to achieve the best results. A 2D visualization (Fig. 5) is created using the Pygame package of the Python environment, which plots the state-space (lanes marked with dots), the planned route, and displays the vehicle.

The environmental model also includes Carla simulator [47] integration, which allows the environment to be displayed in 3D, and can perform several functions. The environmental model is built so that instead of the self-implemented dynamic bicycle model, Carla's four-wheel model can be used for both teaching and evaluation. A simulator function is developed in which the double lane-change solved by the

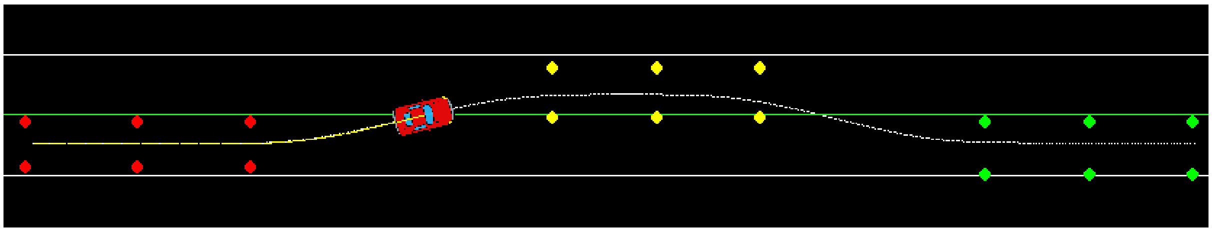


FIGURE 5. 2D rendering.

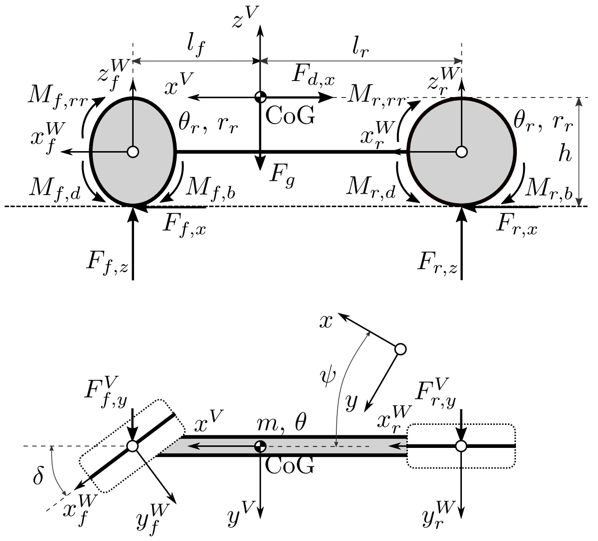


FIGURE 6. Nonlinear single track vehicle model.

agent can be also tried by human drivers with a Logitech G27 racing wheel.

B. SIMULATION OF VEHICLE DYNAMICS

A nonlinear single track vehicle model containing a dynamic wheel model is developed for accurate calculation of the vehicle's motion. An important consideration in designing the model is accuracy in performing high-dynamic driving maneuvers and a balance between computational requirements [12].

The multi-body model (see Fig. 6) consists of three elements, the vehicle chassis, and two virtual wheels, which connected the front and rear axles rigidly. The main parameters are defined according to the vehicle to be modeled, which are the mass m and moment of inertia θ of the chassis, the horizontal distances between the vehicle's CoG and the front and rear wheel centers l_f and l_r, the CoG height of the vehicle h, the moments of inertia $\theta_{[f/r]}$ as well as the radii $r_{[f/r]}$ of the front and rear wheels. The wheel models' parameters also have a strong influence, from which the most important ones are the coefficient of friction $\mu_{[f/r]}$. The Magic Formula defines the transmittable amount of force between road and tires [48]. The slip curves parameters are $C_{[f/r],[x/y]}$, $B_{[f/r],[x/y]}$, $E_{[f/r],[x/y]}$.

C. CLOTHOID BASED PATH GENERATION

The overtaking maneuver consists of a series of straight and curve segments. As shown in Fig. 7, each maneuver is bulit up

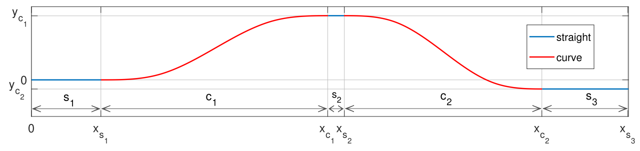


FIGURE 7. The double lane-change path.

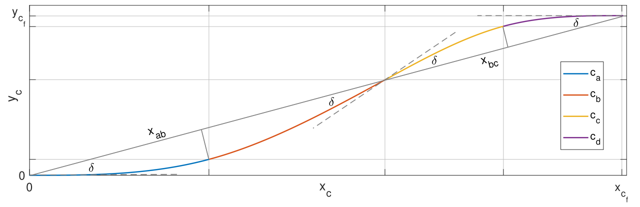


FIGURE 8. Curve section with four primitive clothoid.

of five segments in a fixed order: straight l_1, curve c_1, straight l_2, curve c_2 and straight segment l_3.

The curved segments are composed of clothoid curves that minimize the jerk for ride comfort and safety. The points of the path are evenly distributed along the arc length. The overtaking maneuver path presented in this article uses nine parameters. The s_1, s_2, s_3 specify the length of the straight lines, $c_1, c_2, y_{c_1}, y_{c_2}, p_1, p_2$ specify the curve parameters. A curve shown in Fig. 8 contains four primitive clothoid segments c_a, c_b, c_c, c_d. The clothoid generation solution uses the method from [49], which offers an algorithm to generate a path consisting of symmetric clothoid curves. The sum of the deflection angles of the clothoids is zero $(\delta - \delta - \delta + \delta)$. The method takes as inputs the forward distance x_{c_f}, the lateral distance y_{c_f} and an ratio p parameters. Using these values defined by the action space, formulas (11)-(16) can determine the clothoid segments. The total distance traveled on the curve $s = L$, the curvature κ, the sharpness of the clothoid α. $C(s)$ and $S(s)$ are the Fresnel cosine and sine integrals.

The length of section $[x_{ab}\ y_{ab}]$ is equal to the length of the hypotenuse of the triangle with legs x_{c_f} and y_{c_f}. The parameter p specifies the ratio of x_{ab} and y_{ab}.

$$x_{ab} = p * \frac{\sqrt{x_{c_f}^2 + y_{c_f}^2}}{2},$$
$$x_{cd} = (1-p) * \frac{\sqrt{x_{c_f}^2 + y_{c_f}^2}}{2},$$
$$\delta = \arctan\left(\frac{y_{c_f}}{x_{c_f}}\right) \quad (11)$$

The formulas (12) only show parameters for calculation of section ab. The section is made up of two symmetrical clothoids. The L_{ab} gives the length along the curve, which is divided by two to get the length of the c_a clothoid.

$$L_{ab} = \frac{x_{ab}}{cos_c(\delta)}, \quad L_a = \frac{L_{ab}}{2},$$
$$\kappa_a = \frac{2\delta}{L_a}, \quad \alpha_a = \frac{\kappa_a}{L_a} \tag{12}$$
$$cos_c(\delta) = \begin{cases} \frac{\left(-\sin\delta * C(\eta) + \cos\delta * S(\eta)\right)}{\eta}, & \delta > 0 \\ 1, & \delta = 0 \\ -sin_c(-\delta), & \delta < 0 \end{cases} \tag{13}$$
$$\eta = \sqrt{2|\delta|/\pi} \tag{14}$$

This gives the length of the section along which the Fresnel integrals can be calculated. The sharpness parameter α determines the shape of the curve. By mirroring c_a, c_b can be obtained.

$$x_c(s) = \sqrt{\frac{\pi}{|\alpha|}}\, C\left(\frac{\alpha s}{\sqrt{\pi|\alpha|}}\right), \tag{15}$$
$$y_c(s) = \sqrt{\frac{\pi}{|\alpha|}}\, S\left(\frac{\alpha s}{\sqrt{\pi|\alpha|}}\right) \tag{16}$$

The c_c and c_d are also calculated with this method. By concatenation of the four sections, the entire curve can be determined.

A good solution for estimating the Fressnell sine and cosine with a low absolute error is also provided in [49], which significantly speeds up the trajectory generation process.

D. MODEL PREDICTIVE CONTROL

Model predictive control (MPC), also known as receding horizon control (RHC), is a widely used advanced process control system in automotive applications [50]. This is because MPC can handle problems with many manipulated and control variables even if they are constrained. Hence, MPC can be applied for path tracking of vehicles very easily and successfully. As the name indicates, MPC requires a dynamic model of the controlled process to predict the process's response to the calculated control input. In this case, this is a dynamic vehicle model with two degrees of freedom, which are represented by the lateral vehicle position and the vehicle yaw angle [51]. Besides the mentioned model, the controller uses an optimizer to minimize the user-defined cost function J using the control input, resulting in optimal control. A general example of this cost function is given by (17)

$$J = \sum_{i=1}^{p} W_e e_i^2 + \sum_{i=1}^{p} W_{\Delta u} \Delta u_i^2, \tag{17}$$

where p is the prediction horizon, e_i is the i^{th} error between the reference value and the predicted output, W_e is the weighting coefficient to adjust the relative importance of reference tracking, u_i is the i^{th} manipulated variable and $W_{\Delta u}$ is the weighting coefficient to penalize big changes in u_i. At each timestep, the MPC algorithm calculates the best control input over the control horizon by minimizing the cost function described above and defines the future plant output over the prediction horizon. The actual process is controlled by only the first calculated control input at each step. To reach the desired performance, MPC can be tuned simply by adjusting the weighting coefficients, choosing the right sample time between each prediction, and defining the proper prediction and control horizon.

The MPC controller is part of the classic control loop, with which the reward function determines the value of the reward. The controller was implemented in Python using the Pympc package [52].

IV. RESULTS

The training result is an actor neural network that generates the actions needed to create the path from the state-space and a critic neural network that determines a Q value from the actor's actions and state-space. This Q value approaches the reward value in the case of a learned system, so as a result, it can be highlighted that it gives an estimate of the expected feasibility of the path. As mentioned in Section III, the training phase of the DDPG agent applies noise to the action-space output to retain exploration; therefore, a well-learned agent also produces noise-loaded results. The success of training is determined in the evaluation phase after leaving the noise. The training ends when the reward value converges to a maximum value after successful attempts, and the estimated Q value and reward became closer to each other. The RL agent's training phase consists of nearly 150,000 episodes, and with a powerfully configured computer (i7-9700), it lasted 15 hours and 31 minutes.

From the human point of view, an average driver only gets into a dangerous traffic situation a few times in its life when he or she has to perform an emergency evasive maneuver, and the vehicle comes close to the slip limit. It is usually an unexpected situation. In addition to the vehicle's equipment and technical condition, a lot depends on the driver's individual ability (reflexes, physical condition, fatigue) and driving experience to perform the task without an accident. To evaluate the agent's performance, its results are compared to human drivers performing tests on selected, continuously hardening scenarios. The experiment included ten people with different driving experiences. Some do not have a driving license or even driving expertise in computer games, and on the other end, one of them is the number one pilot on a Formula Student racing team. Some have a few years of driving experience and rarely drive. Most people have 15 to 20 years or more of experience and drive a car daily. Age distribution was between 25 and 42 years. A simulator environment is created to perform the tests, which consisted of the following elements (see Fig. 9):

- Logitech G27 Racing wheel with force feedback
- High-end PC with powerful graphics card (Nvidia 1080Ti)

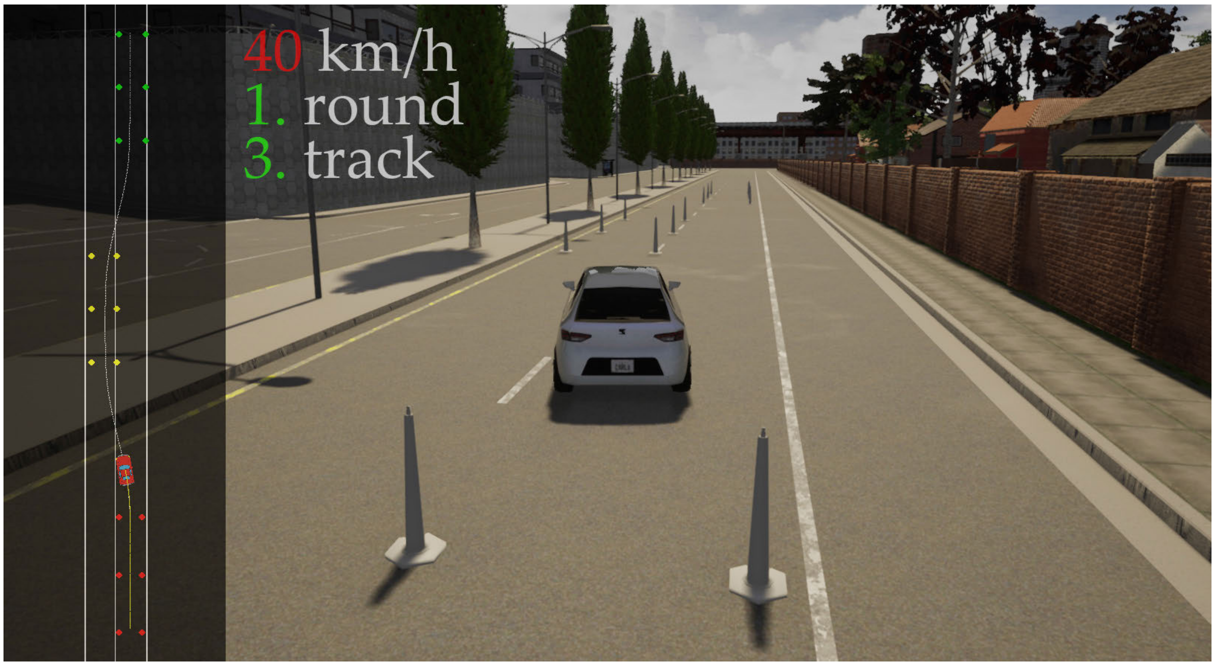


FIGURE 9. The Carla environment used for visual evaluation and for testing with human drivers.

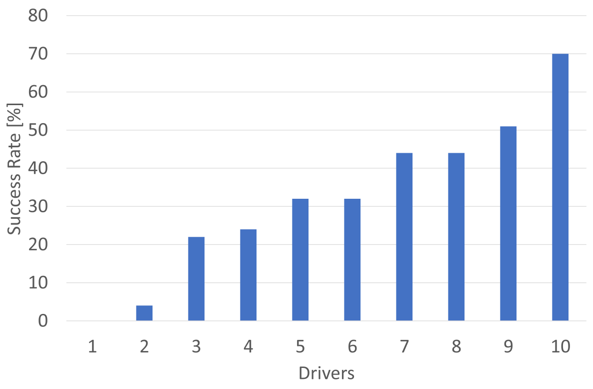


FIGURE 10. Success Rate of each human driver.

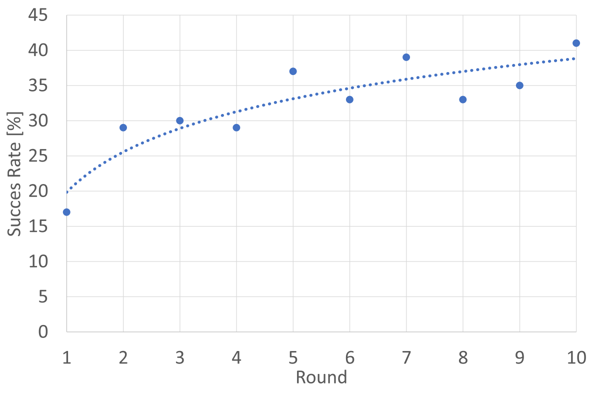


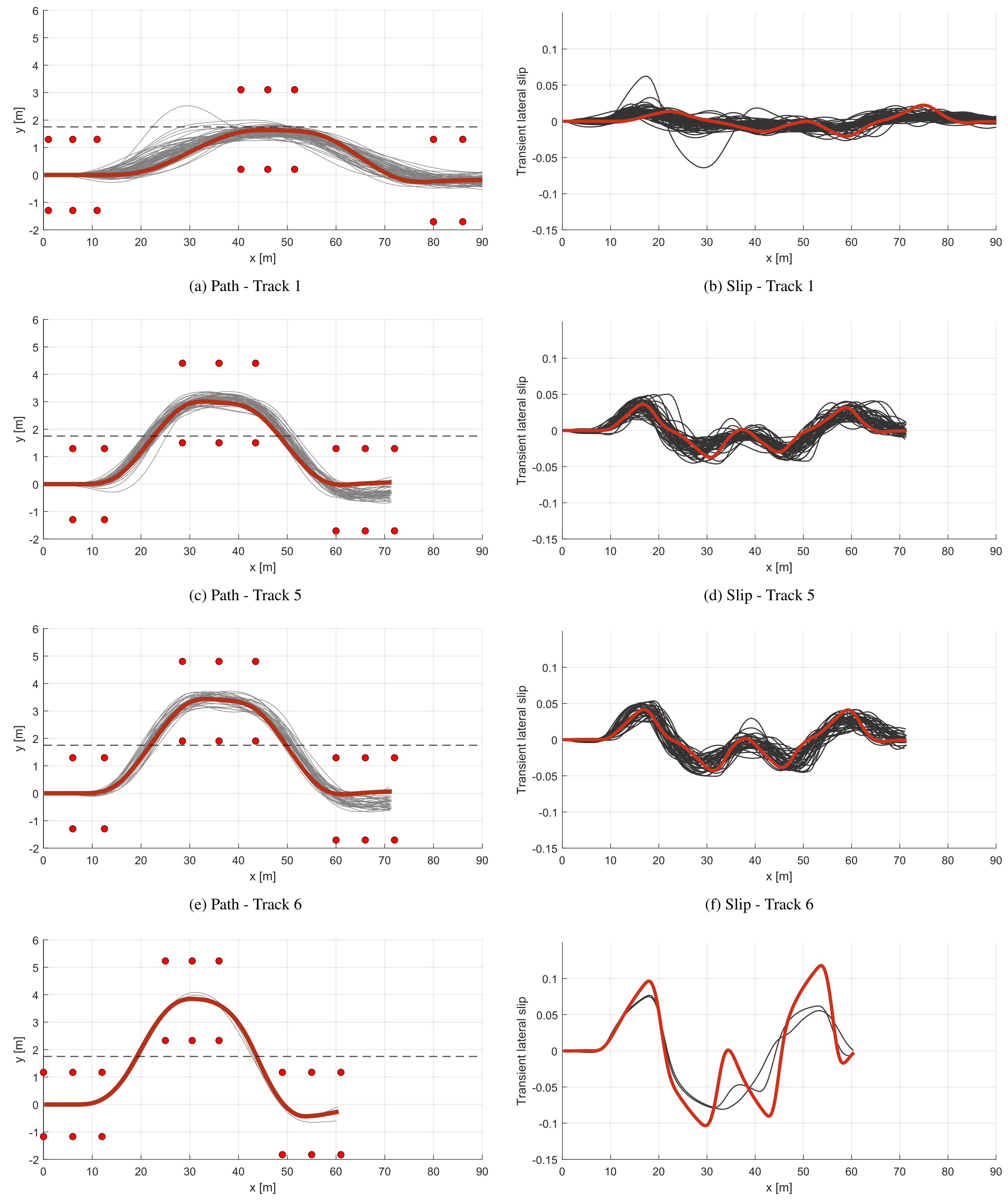
FIGURE 11. Evolution of the human drivers' performance through test rounds.

- CARLA Simulator
- Self-developed Python environment

Each driver had to perform seven randomly generated tracks (tracks no. 1-7) and three moose tests (tracks no. 8-10) at different speeds. The lowest starting speed at the torque release point was 30 km/h, and the highest was 50 km/h. For comparability, everyone had the same random seed. The tracks are 100% successfully fulfilled by the agent, which learned to fulfill all possible tracks in the training range. To vary the scenarios, one round consisted of all tracks, and a driver had to perform ten rounds, which resulted in 100 runs in total/driver. In each case, the driver's job was to pass between the traffic cones in a CARLA environment without leaving the track. When a traffic cone is touched, the thread fails. There was a 5-second waiting time between each attempt until the driver saw the task to be performed (see Fig. 9) on a 3D and top-view 2D display and the initial speed of the vehicle. Same as the original DLC test, only steering was allowed to complete the task. The steering gear was set lower than an average car, which allows for faster steering responses to help drivers. The drivers were allowed to perform a set of seven cycles (7 tries on each of the ten tracks), which were unrecorded and did not appear in the statistics, to help them get familiar with the test environment.

Settings close to reality were selected for the tests. The adhesion factor was chosen to be one, and the parameters of an average vehicle were set. The self-developed vehicle model run during the tests, which did not include Electronic Stability Program (ESP).

Table 2 shows that there is a difference between tracks and people as well. Track 1 was the easiest. It was completed 65 times, though there were some drivers for whom this track



(a) Path - Track 1
(b) Slip - Track 1
(c) Path - Track 5
(d) Slip - Track 5
(e) Path - Track 6
(f) Slip - Track 6
(g) Path - Track 10
(h) Slip - Track 10

FIGURE 12. **Comparison of performances on tracks (1, 5, 6, 10).**

was already way too hard. Track 10 was the hardest, and not surprisingly, only the most experienced racing driver could accomplish it two times. For this statistics, only the fulfilment of the test is evaluated, which is defined successful, when the driver reaches the exit point and the vehicle does not touch any traffic cones.

As expected from the diversity of the drivers, their performance varies widely, as shown in Fig. 10. The worst driver

TABLE 2. Statistical evaluation of the ten drivers' performance on the ten tracks. Each cell shows how many times a driver could reach the exit point of a specific track from 10 tries.

		Tracks										
		1	2	3	4	5	6	7	8	9	10	Sum
Human drivers	1	0	0	0	0	0	0	0	0	0	0	0
	2	1	1	2	0	0	0	0	0	0	0	4
	3	6	7	1	3	3	2	0	0	0	0	22
	4	3	5	0	4	4	2	2	1	3	0	24
	5	10	3	1	8	4	5	0	1	0	0	32
	6	8	8	6	4	2	4	0	0	0	0	32
	7	9	9	4	9	7	1	2	1	2	0	44
	8	9	8	0	6	9	4	3	4	1	0	44
	9	9	9	8	8	8	8	1	0	0	0	51
	10	10	10	8	8	10	10	4	4	4	2	70
	Sum	65	60	30	50	47	36	12	11	10	2	

failed on all tries, and the oldest, though experienced driver, also only succeeded 4%, all from the three easiest tracks. The majority of the drivers performed between 20 and 50%, though most of their successful attempts came from the first six tracks. The racing driver completed 70 attempts, which is almost two deciles higher than the second-best. Though on the last three setups, which are the Moose test defined by the standard on different velocities, his performance was still low (33.3%).

Naturally, the drivers' performance evolved during the test, as shown in Fig. 11. Since the number of tries for each driver at each round is small for creating statistics, their overall learning progress was examined by their cumulated performance in each round. It would also be an exciting experiment, determining how much training a human driver would need, and what is the plateau of human performance. However, on the one hand, this is out of the paper's focus.

For a more in-depth comparison, the diagrams of the lightest (track 1), the most difficult (track 10), and two intermediate (track 5 and 6) tracks are presented in Fig. 12. The figure compares the successful human attempts with the agent's results by showing their actual paths' taken and the rear wheel lateral slip values, which gives a good measure of how well the maneuvers are executed. Gray lines indicate the driver's tracks, while red is the vehicle tracks along the route planned by the agent. All corresponding subfigures' axes are scaled similarly, for the easier comparison of the continuously hardening tasks topological and slip values.

It can be seen in the most comfortable track that it can be accomplished even with big mistakes, and 55 out of the 60 attempts of the best six human drivers are successful. There is even a maneuver on subfigures 12a and 12b, where the human oversteered and corrected before reaching the side lane and still managed to survive. In the middle tracks, successful performances are scattered around the agent's path, but the agent moves on the safest way with a smaller slip value in the middle of the two extremes. Subfigures 12c and 12e show that the agent starts the returning maneuver at the earliest possible instant, before the human drivers, which enables it to stabilize the vehicle earlier in the exit lane, and also reach the centerline of it more precisely, even though in a real emergency situation, this is not that important. In these tries, overall human performance was 41.5%, and for the best six, it was 60%. Only the best driver could successfully drive through the hardest track on the highest speed, in two out of its ten tries (rounds 7 and 10), which is shown in subfigures 12g and 12h. It shows that this track can be completed on almost only one trail. The slip diagrams show that only in the most challenging cases did a slip value greater than 0.1 appear. At this point, the vehicle slipped, but soon came out of the slip and finally passed the moose test at a speed of 50 km/h. According to the drivers, they found that unusually fast steering movements could lead to good results while driving. It was difficult to find the optimum between maneuverability and slip.

V. CONCLUSION

The paper presents an RL based solution for the ISO double lane-change test, in which a TD3 agent generates the geometrical parameters of an evasive maneuver. Afterward, a classic control loop performs direct path tracking. As this is an emergency scenario, the method's practical applicability depends on the algorithm's response time, i.e., the reaction time between recognizing the situation and the generation of the path. For the approach presented in this article, the neural network's response and the geometric planing only need 1ms (PC implementation), which is far in the acceptable range. This shows that for scenes where decision time is crucial, previously trained agents can be an excellent choice for designing highly automated vehicle functions. Naturally, the presented research has its limitations. First, this is a safety-critical vehicle function where the automated driving system should be responsible for its decisions. Though there is an intention in the vehicle industry to standardize the testing, validation, and acceptance of AI-based vehicle functions, it is not available until today. On the other hand, a well-trained agent based on the actor-critic paradigm, like the TD3 presented in this article, provides its results through the actor-network. As a side effect of the learning process, the critic network can give a reasonable estimate of the expected value, e.g., the planned action's feasibility. In this case, this means that if a supervisory system chooses an evasive maneuver, the agent can still provide information on that it is not or slightly feasible. Another limitation of the presented method is that it is trained for a fixed environmental condition parameter set, i.e., the tire friction and other vehicle parameters are constant, and the speed range is limited. On different weather or slippery road conditions, the output is not guaranteed. Though using the method for a broader range of environmental conditions, considering the differences mentioned above, an agent could still learn the appropriate behavior. This article's primary goal is to give a feasibility proof of the presented hierarchical double lane-change maneuver algorithm. It is shown that the presented hierarchical RL-MPC architecture can outperform human drivers and provide a computationally fast feasible solution. In the future, we intend to perform real vehicle tests on an automotive proving ground, under safe conditions, to validate the method.

REFERENCES

[1] Z. Szalay, T. Tettamanti, D. Esztergár-Kiss, I. Varga, and C. Bartolini, "Development of a test track for driverless cars: Vehicle design, track configuration, and liability considerations," *Periodica Polytechnica Transp. Eng.*, vol. 46, no. 1, pp. 29–35, 2018. [Online]. Available: https://pp.bme.hu/tr/article/view/10753

[2] *Passenger Cars—Test Track for a Severe Lane-Change Manoeuvre—Part 2: Obstacle Avoidance*, Standard ISO 3888-2:2011, International Organization for Standardization, Geneva, CH, Standard, 2011.

[3] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Vehicles*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[4] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, May 2020.

[5] H. Vorobieva, N. Minoiu-Enache, S. Glaser, and S. Mammar, "Geometric continuous-curvature path planning for automatic parallel parking," in *Proc. 10th IEEE Int. Conf. Netw., Sens. CONTROL (ICNSC)*, Apr. 2013, pp. 418–423.

[6] X. Li, Z. Sun, Z. He, Q. Zhu, and D. Liu, "A practical trajectory planning framework for autonomous ground vehicles driving in urban environments," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2015, pp. 1160–1166.

[7] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments: Part II," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nice, France, 2008, pp. 1070–1076, doi: 10.1109/IROS.2008.4651124.

[8] M. Likhachev, D. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm," in *Proc. ICAPS*, 2005, pp. 262–271.

[9] M. Montemerlo *et al.*, "Junior: The stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, Sep. 2008.

[10] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1105–1118, Sep. 2009.

[11] R. Pepy, A. Lambert, and H. Mounier, "Path planning using a dynamic vehicle model," in *Proc. 2nd Int. Conf. Inf. Commun. Technol.*, Apr. 2006, pp. 781–786.

[12] F. Hegedüs, T. Bécsi, S. Aradi, and P. Gápár, "Model based trajectory planning for highly automated road vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6958–6964, Jul. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896317318657

[13] C. Bian, G. Yin, L. Xu, and N. Zhang, "Active collision algorithm for autonomous electric vehicles at intersections," *IET Intell. Transp. Syst.*, vol. 13, no. 1, pp. 90–97, Jan. 2019.

[14] Y. Lin, J. McPhee, and N. L. Azad, "Longitudinal dynamic versus kinematic models for car-following control using deep reinforcement learning," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, Oct. 2019, pp. 1504–1510.

[15] E. Alcalá, V. Puig, and J. Quevedo, "LPV-MP planning for autonomous racing vehicles considering obstacles," *Robot. Auto. Syst.*, vol. 124, Feb. 2020, Art. no. 103392.

[16] F. Hegedús, T. Bécsi, S. Aradi, and P. Gáspár, "Motion planning for highly automated road vehicles with a hybrid approach using nonlinear optimization and artificial neural networks," *Strojniski Vestnik J. Mech. Eng.*, pp. 148–160, Mar. 2019.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[18] A. O. Ly and M. A. Akhloufi, "Learning to drive by imitation: An overview of deep behavior cloning methods," *IEEE Trans. Intell. Veh.*, early access, Jun. 15, 2020, doi: 10.1109/TIV.2020.3002505.

[19] A. Folkers, M. Rick, and C. Buskens, "Controlling an autonomous vehicle with deep reinforcement learning," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 2025–2031. [Online]. Available: https://ieeexplore.ieee.org/document/8814124/

[20] J. Lee, T. Kim, and H. J. Kim, "Autonomous lane keeping based on approximate Q-learning," in *Proc. 14th Int. Conf. Ubiquitous Robots Ambient Intell. (URAI)*, Jun. 2017, pp. 402–405. [Online]. Available: http://ieeexplore.ieee.org/document/7992762/

[21] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Hartl, F. Durr, and J. M. Zollner, "Learning how to drive in a real world simulation with deep Q-networks," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 244–250. [Online]. Available: http://ieeexplore.ieee.org/document/7995727/

[22] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in *Proc. 9th Int. Symp. Comput. Intell. Design (ISCID)*, Dec. 2016, pp. 198–201. [Online]. Available: http://ieeexplore.ieee.org/document/7830823/

[23] Y. Ye, X. Zhang, and J. Sun, "Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment," *Transp. Res. C, Emerg. Technol.*, vol. 107, pp. 155–170, Oct. 2019, doi: 10.1016/j.trc.2019.08.011.

[24] J. Wang, Q. Zhang, D. Zhao, and Y. Chen, "Lane change decision-making through deep reinforcement learning with rule-based constraints," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8852110/

[25] P. Wolf, K. Kurzer, T. Wingert, F. Kuhnt, and J. M. Zollner, "Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 993–1000. [Online]. Available: https://ieeexplore.ieee.org/document/8500427/

[26] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 2148–2155.

[27] L. Qian, X. Xu, Y. Zeng, and J. Huang, "Deep, consistent behavioral decision making with planning features for autonomous vehicles," *Electronics*, vol. 8, no. 12, p. 1492, Dec. 2019.

[28] M. P. Ronecker and Y. Zhu, "Deep Q-Network based decision making for autonomous driving," in *Proc. 3rd Int. Conf. Robot. Autom. Sci. (ICRAS)*, Jun. 2019, pp. 154–160. [Online]. Available: https://ieeexplore.ieee.org/document/8808950/

[29] A. Feher, S. Aradi, T. Becsi, P. Gaspar, and Z. Szalay, "Proving ground test of a DDPG-based vehicle trajectory planner," in *Proc. Eur. Control Conf. (ECC)*, May 2020, pp. 332–337.

[30] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," 2020, *arXiv:2001.11231*. [Online]. Available: http://arxiv.org/abs/2001.11231

[31] F. Zeping and D. Jianmin, "Optimal lane change motion of intelligent vehicles based on extended adaptive pseudo-spectral method under uncertain vehicle mass," *Adv. Mech. Eng.*, vol. 9, no. 7, p. 1687814017702823, 2017.

[32] C. Huang, F. Naghdy, and H. Du, "Model predictive control-based lane change control system for an autonomous vehicle," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2016, pp. 3349–3354.

[33] S. Arefnezhad, A. Ghaffari, A. Khodayari, and S. Nosoudi, "Modeling of double lane change maneuver of vehicles," *Int. J. Automot. Technol.*, vol. 19, no. 2, pp. 271–279, Apr. 2018.

[34] J. Lee and H.-J. Chang, "Explicit model predictive control for linear time-variant systems with application to double-lane-change maneuver," *PLoS ONE*, vol. 13, no. 12, Dec. 2018, Art. no. e0208071.

[35] B. Zhou, Y. Wang, G. Yu, and X. Wu, "A lane-change trajectory model from drivers' vision view," *Transp. Res. C, Emerg. Technol.*, vol. 85, pp. 609–627, Dec. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X17302851

[36] X. Ji, K. Yang, X. Na, C. Lv, and Y. Liu, "Shared steering torque control for lane change assistance: A stochastic game-theoretic approach," *IEEE Trans. Ind. Electron.*, vol. 66, no. 4, pp. 3093–3105, Apr. 2019.

[37] K. Yang, R. Zheng, X. Ji, Y. Nishimura, and K. Ando, "Application of stackelberg game theory for shared steering torque control in lane change maneuver," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 138–143.

[38] F. Roselli, M. Corno, S. M. Savaresi, M. Giorelli, D. Azzolini, A. Irilli, and G. Panzani, "H_∞ control with look-ahead for lane keeping in autonomous vehicles," in *Proc. IEEE Conf. Control Technol. Appl. (CCTA)*, Aug. 2017, pp. 2220–2225.

[39] Y. Tian, Y. Zhao, Y. Shi, X. Cao, and D.-L. Yu, "The indirect shared steering control under double loop structure of driver and automation," *IEEE/CAA J. Automatica Sinica*, early access, Jan. 16, 2020, doi: 10.1109/JAS.2020.1003018.

[40] A. Fehér, S. Aradi, F. Hegedűs, T. Bécsi, and P. Gáspár, "Hybrid DDPG approach for vehicle motion planning," in *Proc. 16th Int. Conf. Informat. Control, Autom. Robot., ICINCO*, vol. 1, O. Gusikhin, K. Madani, J. Zaytoon, Setúbal, Portugália: SciTePress, 2019, pp. 422–429.

[41] I. Caspi, G. Leibovich, G. Novik, and S. Endrawis, "Reinforcement learning coach," Tech. Rep., Dec. 2017, doi: 10.5281/zenodo.1134899.
[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, 2014, pp. 1–9.
[43] S. Ravichandiran, S. Saito, and R. Shanmugamani, *Python Reinforcement Learning*. Birmingham, U.K.: Packt Publishing, 2019. [Online]. Available:https://books.google.hu/books?id=US4HxQEACAAJ
[44] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[45] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, p. 823, Sep. 1930.
[46] H. Lengyel, T. Tettamanti, and Z. Szalay, "Conflicts of automated driving with conventional traffic infrastructure," *IEEE Access*, vol. 8, pp. 163280–163297, 2020.
[47] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," Nov. 2017, *arXiv:1711.03938*. [Online]. Available: https://arxiv.org/abs/1711.03938
[48] H. B. Pacejka, "Applications of transient tire models," in *Tire and Vehicle Dynamics*, 3rd ed., H. B. Pacejka, Ed. Oxford, U.K.: Butterworth-Heinemann, 2012, pp. 355–401.
[49] D. K. Wilde, "Computing clothoid segments for trajectory generation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2009, pp. 2440–2445. [Online]. Available: http://ieeexplore.ieee.org/document/5354700/
[50] B. Mehta and Y. Reddy, "Advanced process control systems," in *Industrial Process Automation Systems*, B. Mehta and Y. Reddy, Eds. Oxford, U.K.: Butterworth-Heinemann, 2015, pp. 547–557. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012800939000019X
[51] R. Rajamani, *Vehicle Dynamics and Control*. Boston, MA, USA: Springer, 2012, doi: 10.1007/978-1-4614-1433-9.
[52] M. Forgione, D. Piga, and A. Bemporad, "Efficient calibration of embedded MPC," in *Proc. 21st IFAC World Congr.*, Berlin, Germany, Jul. 2020, pp. 5263–5268.

ÁRPÁD FEHÉR received the B.Sc. and M.Sc. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the Department of Control for Transportation and Vehicle Systems.

His research interests include vehicle dynamics, mechatronics, machine learning, and especially reinforcement learning.

SZILÁRD ARADI (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2005 and 2015, respectively.

He is currently working with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, where he has been a Senior Lecturer, since 2016. His research and industrial works have involved railway information systems, vehicle on-board networks, and vehicle control. His research interests include embedded systems, communication networks, vehicle mechatronics, and reinforcement learning.

TAMÁS BÉCSI (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics, Budapest, Hungary, in 2002 and 2008, respectively.

Since 2005, he has been an Assistant Lecturer with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, where he has been an Associate Professor, since 2014. His research and industrial works have involved railway information systems, vehicle control, and reinforcement learning. His research interests include linear systems, embedded systems, traffic modeling, and simulation.