applied sciences

MDPI

Article

Design of a Reinforcement Learning-Based Lane Keeping Planning Agent for Automated Vehicles

Bálint Kővári, Ferenc Hegedüs and Tamás Bécsi *

Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, H-1111 Budapest, Hungary; balint.kovari@edu.bme.hu (B.K.); hegedus.ferenc@mail.bme.hu (F.H.)
* Correspondence: becsi.tamas@mail.bme.hu

Received: 17 September 2020; Accepted: 9 October 2020; Published: 14 October 2020

Featured Application: The presented method can be used as a real-time trajectory following algorithm for autonomous vehicles using prediction based on lookahead information.

Abstract: Reinforcement learning-based approaches are widely studied in the literature for solving different control tasks for Connected and Autonomous Vehicles, from which this paper deals with the problem of lateral control of a dynamic nonlinear vehicle model, performing the task of lane-keeping. In this area, the appropriate formulation of the goals and environment information is crucial, for which the research outlines the importance of lookahead information, enabling to accomplish maneuvers with complex trajectories. Another critical part is the real-time manner of the problem. On the one hand, optimization or search based methods, such as the presented Monte Carlo Tree Search method, can solve the problem with the trade-off of high numerical complexity. On the other hand, single Reinforcement Learning agents struggle to learn these tasks with high performance, though they have the advantage that after the training process, they can operate in a real-time manner. Two planning agent structures are proposed in the paper to resolve this duality, where the machine learning agents aid the tree search algorithm. As a result, the combined solution provides high performance and low computational needs.

Keywords: reinforcement learning; autonomous vehicle; vehicle dynamics; lane keeping; planning agents; Q-learning; policy gradient; Monte Carlo Tree Search

1. Introduction

Deep Learning (DL) has gained tremendous interest in the field of vehicle control and motion planning in general thanks to the success of DL in other fields where the advantages of Convolution Neural Networks (CNN) are heavily utilized such as semantic segmentation, scene understanding, object detection, and recognition [1,2]. The early concepts for designing autonomous functions used rule-based systems where the parameters of the conditions are tuned over tests. This type of system has limitations because it is hard to cover all the rare events in tests that can happen in the real world. At the same time, Deep Learning techniques feature adaptation and self-tuning. Consequently, these techniques are able to generalize behaviors for unseen cases. These characteristics make DL an obvious choice for solving vehicle control problems. The vehicle control problem can be categorized into three main groups. The first is lateral motion control of the vehicle, which is dedicated to managing the in-lane location of the vehicle and also handling maneuvers that require lateral control such as lane-changing. The second is a longitudinal motion control, which is dedicated to managing the brake and throttle pedals to hold speed, thus maintaining

the appropriate following distance. The third group is when the lateral and longitudinal motion have to be controlled simultaneously. For a more comprehensive study on the field of DL based vehicle control, see in [3,4].

The authors in the Autonomous Land Vehicle in a Neural Network (ALVINN) [5,6] project endeavor to utilize supervised learning (SL), where the input is raw camera data (30×32), and the neural network only has one hidden layer with four neurons, while the output is one discrete steering angle out of thirty. The training dataset is enlarged with data augmentation, which also decreased the occurrence of overfitting. After the training, the real-word test shows that the trained neural network is able to hold the vehicle on the centerline of the lane more precisely than the human driver could. The following significant result of SL in the vehicle control field is the DARPA Autonomous Vehicle (DAVE) project [7,8]. It has to be mentioned that the innovation in computational hardware has a significant role in the success of this project because it uses a CNN that has 250,000 tunable parameters. The training data is collected from a human driver with cameras implemented to the vehicle. For the steering of the vehicle, an end-to-end solution is carried out with the help of a CNN. The trained CNN is able to detect the main signs that are painted on the road. Moreover, it can perform lane-keeping in a way when it does not need any human intervention. Another remarkable result is in [9], where the authors utilize CarSim [10] for data generation, which is trained with various optimizers after filtering out the degenerated samples from the dataset. The trained neural networks are only evaluated in the same simulator. The results showed that the steering signal created by the neural network is noisy, which suggests the use of Recurrent Neural Networks that has a memory of earlier inputs. By utilizing RNNs, the neural network can understand being in a maneuver, which can result in a smooth steering signal. In [11], the authors train a Convolution Long Short-Term Memory Neural Network (C-LSTM). In training, camera data are used with time dependencies. The performance of the trained network compared to another network trained with simple CNN [12] shows that the C-LSTM reaches a higher performance level in steering the vehicle.

The first reinforcement learning (RL) based control is inspired by the concept of ALVINN [13]. The output of the neural network is discrete steering angles as in ALVINN. It has an essential advantage over the SL-based concepts, which is the ability to learn in an online manner from experiences gathered through interactions with the environment. The architecture is a simple feedforward network in which raw camera data are used as input. A better concept is proposed in [14], which uses the well-known Deep Q-network algorithm for training the neural network to predict the appropriate yaw acceleration for a lane-changing maneuver. Most of the longitudinal control problems are complex and show firm nonlinear behavior, which makes deep learning-based solutions suitable for this type of control task because these methods support adaptation and demonstrate the capability of handle nonlinear control problems. The authors of [15] combine fuzzy logic with reinforcement learning and create a hybrid method for the longitudinal control of a vehicle. The RL method is a Q estimator network, and for fuzzy inference they use Takagi–Sugeno. The reward function only uses the distance between the ego and the front vehicle, which is not able to provide a solution that considers safety and efficiency aspects in the decision-making. Consequently, in such control problems, a multi-objective reward scheme is required for appropriate control. In [16], a Policy Gradient-based agent is proposed, which uses a multi-objective reward function to train the agent for the control task of Cooperative Adaptive Cruise Control. In the reward function, the time distance between the ego and the front vehicle is used along with the derivative of this quantity. The actions, in this case, are simple brake, throttle, and pass commands. The trained network is tested in simulation and the results show that it can keep the required time distance accurately, but the control signal always oscillates, which is uncomfortable for the passengers. This behavior can be corrected by sanctioning this exact operation during training by the reward function, but another approach is the use of RNN networks. The authors in [17] propose a Parametrized Batch Actor–Critic algorithm that utilizes both the policy and value-based model-free RL techniques for a longitudinal control problem. In this case, the real-word tests

show that the multi-objective reward system mitigates the oscillation, which results in great comfort for passengers. In [18], the authors introduce a Deep Q-Learning (DQN)-based method that is dedicated to controlling the braking system for avoiding collisions autonomously. Along with the original memory buffer of DQN, they use a special memory where only experiences of critical events are stored. These types of events are rare, so without this memory, these events become underrepresented, which deteriorates the reliability of the agent. The trained model is tested in situations where accidents have to be avoided with pedestrians. The model reaches excellent performance, and it is also tested according to the restrictions of the Euro NCAP test protocol, and the model passes this test too. The methods above utilize reinforcement learning's unique features such as online learning; however, it also has some disadvantages like the sample inefficiency. To mitigate these issues, several authors propose methods that combine reinforcement and supervised learning approaches, where supervised learning accelerates the training process, but thanks to reinforcement learning, the possibility of adaption is maintained. For such concepts see in [19,20].

Autonomous driving requires both lateral and longitudinal control to operate in a simultaneous manner to perform the dynamic driving task on a proper performance level [21]. A method that combines supervised learning and reinforcement learning is presented in [22], where the authors use the exact idea for formulating their own algorithm to solve the lateral and longitudinal control task simultaneously, which control task is formulated as keeping the vehicle on the road. In this particular algorithm, DQN is combined with SL, where the experiences are gathered from professional drivers to accelerate training. The DQN's experience memory is also filtered to eliminate poor experiences. In [23], the authors compare the DQN and the Deep Deterministic Actor–Critic (DDAC) algorithm to show how important the usage of continuous actions in such control problems. After training, the models are evaluated in The Open Racing Car Simulator (TORCS), and the DQN reaches a secondary performance level thanks to the fact that it does not support continuous actions. At the same time, the DDAC supports continuous actions, and it provides superior performance compared to DQN. In [24], a planning feature-based, deep behavior decision method trained with TD3 was proposed to select an optimal maneuver for autonomous vehicles in dynamic traffic. The authors of [25] propose a CNN-based method for the same autonomous braking system concept as introduced earlier, but in that case, the model is capable of managing both steering and acceleration. This concept utilizes a Variational AutoEncoder (VAE) combined with RNN to predict the position alteration of obstacles and uses the Deep Deterministic Policy Gradient (DDPG) for learning the control policy. The model is evaluated in simulation, and the results show that it is capable of decreasing the ration of collision by 60%. In the realm of model-free reinforcement learning algorithms, a critical part is to specify the reward function for the task, because this feedback signal is the only thing that can lead the training of the model to success. Unfortunately, in such complex domains, it is tough to formulate the correct reward function. To solve that problem, Inverse Reinforcement Learning (IRL) is also utilized, which is a subfield of RL, where the algorithm tries to develop a reward function by monitoring an expert interacting with the environment. In [26], the authors presented a concept that utilizes IRL, and they manage to train the agent to learn different driving styles from a presentation. Moreover, a good review of imitation-based and behavior cloning methods applied in this field can be found in [27].

Contributions of the Paper

The paper deals with the design of a lateral control agent and proposes two new algorithms that utilize planning after learning in the prediction phase that incorporates the real-time applicability of the neural networks with the robustness of the Monte Carlo tree search (MCTS), which can overcome both algorithms' limitations. Moreover, this paper presents a unique implementation of the MCTS that enables the utilization of such a tree search algorithm in a computationally exhausting control task like that. It also compares the performance of both the original and the new MCTS concept. Furthermore, the paper reveals

the importance of the lookahead information in the state representation by comparing agents trained with and without it. Finally, the paper presents a detailed statistic and strategic comparison of all the introduced approaches for this control task to assess and understand the different algorithms' advantages and disadvantages.

2. Environment

The formulation of the environment explicitly affects the level of robustness that the agents can reach. Consequently, the goals of the environment have to cover the diversity of the trajectories, therefore a wide variety of turn combinations have to be created along the training process. The trajectory generator realizes the required versatility of the trajectories through a parametrizable random process that provides a new trajectory for every episode. The random process, which results in different trajectories, starts with choosing the diameter of a circle, which is followed by choosing the number of holding points that are placed on the arc evenly. In the following step, the positions of the holding points are changed in their local neighborhood randomly. At the end of this process, a random trajectory is created by its holding points by fitting a spline to the holding points. The created spline is complemented with two other splines that are interpreted as the edges of a lane while the original spline is the centerline. The edges enable the customization of the lane-keeping problem through the lane width, and it is also advantageous in the visualization of the control problem. Figure 1 shows some example trajectories.

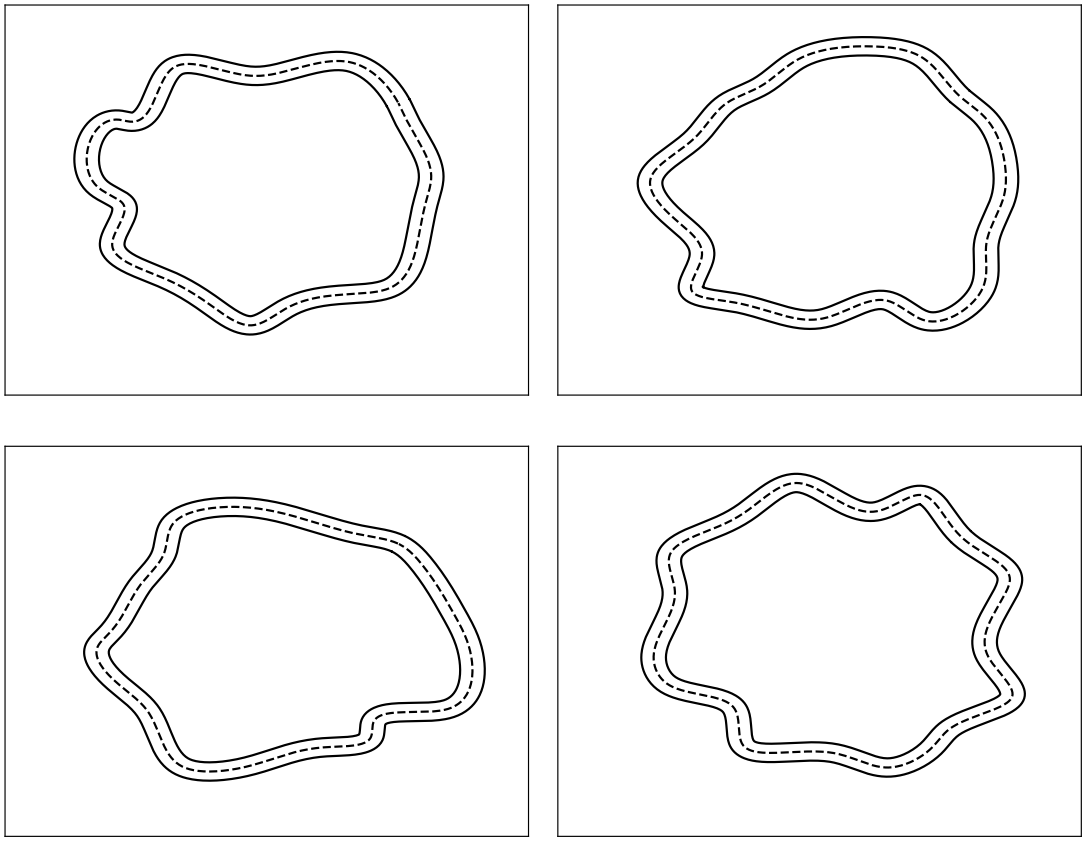


Figure 1. Examples of randomly generated trajectories for training.

The representation of the control task is also a necessity as the state vector is the only descriptor that details the problem for the agent. Accordingly, the sensor information is decomposed into two parts because the representation also has two primary intentions. On the one hand, the actual position of the vehicle in the lane is essential, while on the other hand, the agent has to know how the lane evolves over a limited horizon based on the current state of the vehicle.

These two components are equally important because precise trajectory tracking requires that every decision has to be adequate both for the current situation and for the trajectory in front of the vehicle. The current state of the vehicle is detailed with the help of two pieces of information. The first is the in-lane position and the second is the relative yaw angle, both transformed into $[-1, 1]$ interval. The relative yaw angle shows the deviation of the vehicle heading from the tangent of the given point of the trajectory. The part of the trajectory which is in front of the vehicle is detailed by relative yaw angles calculated into equally distanced points of the horizon with respect to the current state of the vehicle. This approach makes this part of the sensor information easily customizable. Thus, the representation can be easily changed in the training procedure to find the suitable one. A few configurations are shown in Figure 2. The sensor information vector is shown in Table 1.

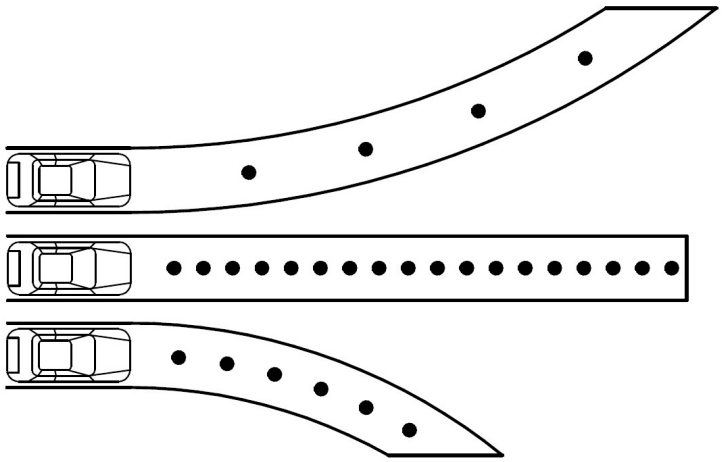


Figure 2. Different configurations of the lookahead sensor information part.

Table 1. A configuration of the state vector.

Position	**Relative Yaw**	**Relative Yaw**	**Relative Yaw**	**Relative Yaw**	**Relative Yaw**	**Relative Yaw**
d	φ_0	φ_1	φ_2	φ_3	φ_4	φ_5
Actual state of the vehicle		Lookahead information about the trajectory ahead				

The elements in the sensor information vector that are used for the representation of the problem are relative quantities. Thus, this information only depends on the relation between the vehicle and the lane. This approach enables the agent to learn all the trajectories that can be constructed from the curvatures of the turns created during training. This concept gains additional robustness over the approach that learns only the given trajectories thanks to the influence of the absolute quantities. This can be seen as an implicit acceleration of the training process. Moreover, the relative quantities help the agent in generalization, which is a crucial point in the success of the training.

2.1. Vehicle Model

The last sensitive aspect of the environment is the choice of the vehicle model, thanks to the controversy between maneuver accuracy and computational requirements. Choosing the appropriate complexity for a vehicle model is always a challenge because two different goals have to be reached at the same time. On the one hand, the vehicle model should be as realistic as possible to cover all possible driving situations, while on the other hand, the computational need of the model has to be scaled back as much as possible to decrease the runtime of the final simulation [28]. The used vehicle model is a nonlinear single track model complemented with a dynamic wheel model. This vehicle model represents an excellent trade-off between computational efficiency and vehicle dynamics because it provides fundamental dynamics for a wide variety of maneuvers. Figure 3 shows the multi-body model of the vehicle; it also shows that the main parts of the model are the vehicle chassis and two rigidly attached wheels that also symbolize the axles of the front and the rear. The model has several parameters; thus to provide a transparent description, all the influential parameters are detailed in Table 2.

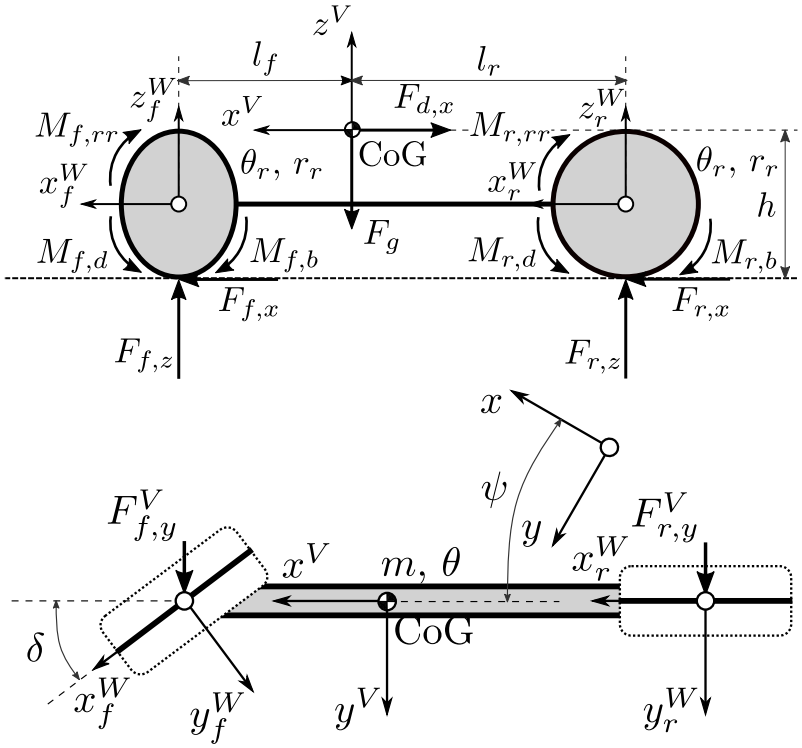


Figure 3. Sideview and top view of the single track nonlinear model [29].

Table 2. Parameters of the vehicle model.

Parameter	Meaning
m	Mass
θ	The inertia of the chassis.
l_f	The horizontal distance between the center of the front wheel and the vehicle's center of gravity.
l_r	The horizontal distance between the center of the rear wheel and the vehicle's center of gravity.
h	The height of the vehicle's center of gravity.
$\theta_{f/r}$	The moments of inertia of the front and rear wheel.
$r_{f/r}$	The radius of the front and rear wheel.

It is also necessary to introduce the influential parameters of the wheel model, which are the Magic formula's $C_{[f/r],[x/y]}$,$B_{[f/r],[x/y]}$,$E_{[f/r],[x/y]}$ parameters, which are fitting constants of a specific wheel–ground contact pair. These determine how much force can be transmitted between the tire and the road surface [30] based on the slip value. Moreover, the friction coefficient also has to be mentioned as a relevant parameter of the wheel model.

The model has three inputs, which are the breaking M_b and driving M_d torques applied at the wheels, and the third is the steering angle δ of the front wheel. The distribution of the brake torque $M_{[f/r],b}$ is ideal for providing equal brake slips, while the driving torque $M_{[f/r],d}$ is distributed between the front and rear with a time-dependent distribution factor ξ_M. In the following model description, the variables without a subscription have to be considered in a ground-fixed coordinate system, while the variables with subscription W have to be considered in the wheel-fixed coordinate system and variables with subscription V have to be considered in a vehicle-fixed coordinate system, and the dot notation represents time derivatives. The possible direction of movements of the chassis is the movement along the x longitudinal axis, the y lateral axis, and the rotation around the chassis Ψ vertical axis, while the wheels are only able to rotate around their horizontal axes $\phi_{[f/r]}$, and the lateral and longitudinal slips modeled in a dynamical manner $s_{[f,r],[x,y]}$. The equations of the chassis are based on Newton's second law:

$$\ddot{x} = \frac{1}{m}(F_{f,x} + F_{r,x} + F_{d,x}) \tag{1}$$

$$\ddot{y} = \frac{1}{m}(F_{f,y} + F_{r,y} + F_{d,y}) \tag{2}$$

$$\ddot{\Psi} = \frac{1}{\theta}(l_f F^V_{l,f} - l_r F^V_{r,y}) \tag{3}$$

where the $F_{[f/r],[x,y]}$ variables are the tire forces of the vehicle, the forces derived from aerodynamic drag is modeled and calculated as follows,

$$F^V_{d,x} = \frac{1}{2} c_D A_f \rho_A \dot{x}^V \sqrt{(\dot{x}^V)^2 + (\dot{y}^V)^2} \tag{4}$$

$$F^V_{d,y} = \frac{1}{2} c_D A_f \rho_A \dot{y}^V \sqrt{(\dot{x}^V)^2 + (\dot{y}^V)^2} \tag{5}$$

where ρ_A is the density of the air, c_D is the coefficient of drag, and A_f is the size of the frontal surface of the vehicle.

Only the equations of the front wheel are presented because the rear and front are modeled identically, the forces of the tire are derived from the motion of the wheel, and the equations are formalized by taking into consideration Newton's second law and the dynamic slip equations from the work in [30]:

$$\ddot{\phi} = \frac{1}{\theta_f}(M_{f,d} - r_f F^W_{f,x} - M_{f,b} - M_{f,rr}) \tag{6}$$

$$\dot{s}_{f,x} = \frac{1}{l_{f,x}}(r_f \dot{\phi}_f - \dot{x}^W_f - |\dot{x}^W_f| s_{f,x}) \tag{7}$$

$$\dot{s}_{f,y} = \frac{1}{l_{f,y}}(-\dot{y}^W_f - |\dot{y}^W_f| s_{f,y}) \tag{8}$$

The variable $\dot{x}_f^W$ is the longitudinal velocity of the center of the wheel, while the $\dot{y}_f^W$ is the lateral velocity of the same wheel center. The relaxation lengths which depend on the lateral and longitudinal slips are calculated as follows,

$$l_{f,[x/y]} = \max\left(l_{f,[x/y],0}\left[1 - \frac{B_{f,[x/y]}C_{f,[x/y]}}{3}|s_{f,[x/y]}|\right], l_{f,[x/y],min}\right) \tag{9}$$

The $l_{f,[x/y],0}$ represents the values at wheel lock or wheel spin, while $l_{f,[x/y],min}$ notes the value for the standstill case. The SAE J2452 standard is used to calculate the torque of the rolling resistance M_{f_rr}, and the Magic formula provides the computational model for the calculation of the tire forces, which looks as follows,

$$\begin{aligned}\tilde{F}^W_{f,[x/y]} =& \mu_f F^W_{f,z} \sin\{C_{f,[x/y]} \arctan(B_{f,[x/y]}\tilde{s}_{f,[x/y]} - \\ & E[B_{f,[x/y]}\tilde{s}_{f,[x/y]} - \arctan(B_{f,[x/y]}\tilde{s}_{f,[x/y]})])\}.\end{aligned} \tag{10}$$

To enhance and maintain the stability of the numerical computational process, dumped slip values are used in the calculation of the forces, which looks as follows,

$$\tilde{s}_{f,x} = s_{f,x} + \frac{k_{f,x}}{B_{f,x}C_{f,x}\mu_f F^W_{f,z}}(r_f\dot{\rho}_f - \dot{x}^W_f) \tag{11}$$

$$\tilde{s}_{f,y} = s_{f,y} \tag{12}$$

where $k_{f,x}$ is the factor that depends on the velocity, and it is calculated as follows,

$$k_{f,x} = \begin{cases} \frac{1}{2}k_{f,x,0}(1 + \cos(\pi\frac{|\dot{x}^W_f|}{v_{low}})) & \text{if } \dot{x}^W_f \leq v_{low} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where the damping factor $k_{f,x,0}$ belongs to the zero velocity case, and the damping factor is switched off for the v_{low} case. The friction ellipse method is used to calculate the superposition of the forces, which is given as follows,

$$F^W_{f,x} = sign(\tilde{s}_{f,x})\sqrt{\frac{(\tilde{F}^W_{f,x}\tilde{F}^W_{f,y})^2}{(\tilde{F}^W_{f,y})^2 + (\frac{\tilde{s}_{f,y}}{\tilde{s}_{f,x}}\tilde{F}^W_{f,x})^2}} \tag{14}$$

$$F^W_{f,y} = sign(\tilde{s}_{f,y})\sqrt{\frac{(\tilde{F}^W_{f,x}\tilde{F}^W_{f,y})^2}{(\tilde{F}^W_{f,x})^2 + (\frac{\tilde{s}_{f,x}}{\tilde{s}_{f,y}}\tilde{F}^W_{f,y})^2}} \tag{15}$$

An explicit Ordinary Differential Equation solver is used with a step size of approximately 1 ms for the introduced wheel model. To further enhance the computational efficiency of the vehicle model, it is implemented in C instead of python. For more details about the vehicle model see the works in [29,31].

2.1.1. Action Space

The action space is discrete and consists of steering angles only. Therefore, the velocity of the vehicle is fixed. A simple PD controller provides the torque commands to keep the velocity of the vehicle. It is critical to choose the number of actions appropriately, as this is also the branching factor of the tree search, and this parameter hugely influences the complexity of the search problem. Moreover, the quality of the

reachable trajectory tracking is also affected by the action space, but both the number of actions and the actual values of the steering angles are influential. The reward scheme for the trajectory tracking problem is detailed in Equation (16).

$$R = \begin{cases} \cos\phi_0 + |d| & \text{if } R > 0 \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

The expression incorporates two essential performance measures into a single reward function, notably the agent gets the maximum possible reward if the vehicle stays on the trajectory, i.e., in the middle of the lane, while its relative yaw angle is equal to zero. Furthermore, it is important to mention that the environment works with discrete time steps that are customizable.

3. Methodology

3.1. Monte Carlo Tree Search

The most concerning weakness of classic uninformed search methods is not that they cannot find the optimal solution if there is one, but that they cannot operate in a real-time manner because the needed computational power is lacking. On the other hand, greedy algorithms that use heuristics to find optimal solution come with no guarantees. Thus, they likely end up in suboptimal solutions or with no solution at all. The Monte Carlo tree search algorithm mitigates these concerns thanks to its unique approach to the node selection procedure and individual nodes' value assignment. The Upper Confidence bound applied for Trees (UCT) algorithm is responsible for selecting the currently most promising nodes in every branch during the iteration process. Therefore, a bandit algorithm navigates through the tree from root to leaf, which handles the exploration-exploitation dilemma as a tunable trade-off. Consequently, the heart of MCTS is the bandit algorithm. The bandit-based approach is firstly introduced in the realm of clinical trials [32], and the research area of animal and human learning [33]. Such methods can be applied to any decision-making problem that has uncertainty. For more details on bandit algorithms, see [34]. MCTS algorithm's value assignment is unique since it determines a node's value by continuing the process from the given state until a terminal state, with a default policy. Random play based value assignment seems insufficient at first, but research papers that apply parallelization to this algorithm show that the averaged value of more random play from the given node does not result in a meaningful performance boost [35]. Thus a single random rollout is quite representative. The UCT algorithm looks as follows,

$$\overline{X_i} + 2C_p\sqrt{\frac{2\ln N_i}{n_i}}, \tag{17}$$

where $\overline{X_i}$ is the average value of the currently examined node, C_p is a constant that helps in the controlling of the exploitation–exploration trade-off, and N_i is the number of visits of the ancestor node, while n_i is the number of visits of the currently examined node. The MCTS algorithm has an essential advantage over its competitors, notably that it converges to the globally optimal solution if enough time is given [36]. For a more comprehensive study on MCTS, see in [37].

This approach can be seen as a parametrizable trade-off between the uninformed and the greedy search concepts. The first part of the UCT algorithm represents exploitation. Therefore, if it chooses the next node exclusively, it behaves as a greedy algorithm. On the other hand, the second part represents exploration that leads to the node, which has the lowest visit count on the branch. Thus if it chooses the next node exclusively, it behaves like a uninformed search algorithm.

The MCTS algorithm builds a tree-based representation of the domain by repeating the Selection, Expand, Rollout, and Backpropagation steps.

- Selection: Selects the nodes with the highest UCT value in every selection process until a leaf node is found.
- Expand: If the examined node has not visited yet, it populates its child nodes, if there is any.
- Simulation: Executes a Monte Carlo rollout until a terminal state is found.
- Backpropagation: If a terminal state is found at the end of the simulation, the terminal state's value is backpropagated along the path from the leaf to root.

After a certain number of iterations, the MCTS suggests an action. Thus, the action selection policy is essential. In this case, the robust-child [37] approach is utilized, which selects the action that instantiates the child node that has the most visits from all child nodes on the branch.

In this paper, the MCTS algorithm is utilized as a benchmark for comparison to other algorithms and a training data generator for supervised learning.

3.2. Supervised Learning

Supervised learning is a field of machine learning where the goal is to tune a function approximator to reestablish the connection between input vector x and output label y based on the provided labeled dataset. The reservations about SL are mostly attached to its insatiable need for data, which can be critical in several fields because the required amount of data can not be provided. Creating datasets for training is always a resource-intensive task, but in this case, the MCTS algorithm is used as a training sample generator for SL, as it can solve the control problem on a high-performance level. Furthermore, the MCTS, combined with the introduced environment that randomly creates trajectories, is a perfect fit for such an application because it can generate as many training samples as required. SL's dependence on labeled data shows a theoretical limitation to SL, notably that it can only resolve previously solved problems. Still, the real-time applicability and generalization feature of neural networks that enable the trained network to operate in a wilder interval than presented through the training samples is undoubtedly beneficial.

3.3. Reinforcement Learning

By investigating the nature of learning and intelligence, it occurs that humans organize experiences into a causal structure to make decisions based on the information gathered and ordered to achieve some desired goal or behavior. Therefore, the interactions with the environment always help understand the world and develop behavior needed for several purposes. This is the exact concept of RL. The tools of intelligence that serve humans in problem-solving can be found in the RL framework. These tools are the capability to think on an abstract level, the sensitivity for finding the context's algorithm, and the capability to recognize analogies and patterns in different scenarios. The formulation of the state descriptor representing the control task for the agent accomplishes the same as thinking on an abstract level as it separates the influential features of the environment from the irrelevant. The reward function and the utilized credit assignment concept realize the algorithm's functionality, which could be immediate or discounted; these concepts determine how the individual slices of time steps affect the outcome. The recognition of analogies is done by the neural network, which tries to generalize the experiences to behave in the right fashion in unseen scenarios.

The model-free RL framework is formalized as a Markov Decision Process (MDP) $\{S, A, T, R\}$. The agent tries to form an optimal behavior without prior knowledge, only through interactions with the environment in an online manner. During the interactions, the agent changes the state of the environment by triggering the execution of actions. The environment characterizes the immediate consequences of the chosen action, which can be a reward or a punishment, depending on the scalar feedback sign. Therefore,

the agent's goal is to maximize the cumulative feedback for the whole process called an episode, which can be formalized as follows,

$$G = \sum_{t=1}^{T} \gamma^t r_t. \tag{18}$$

The training loop of RL is shown in Figure 4. For a more comprehensive study on reinforcement learning, see in [38].

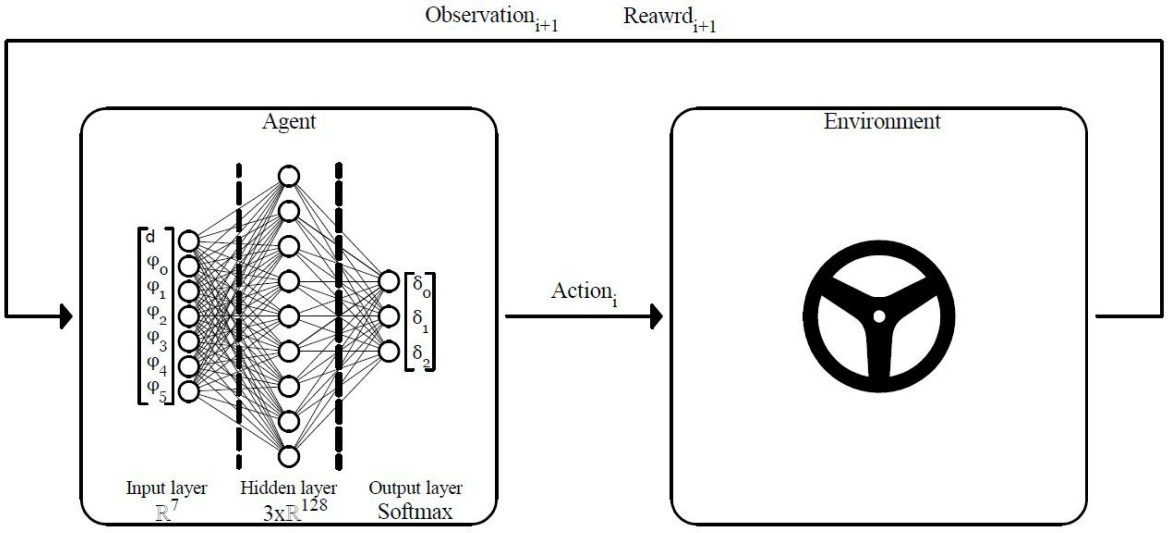


Figure 4. The RL training loop.

3.4. Value-Based

The first profound breakthroughs of RL are attached to the value-based DQN algorithm, which showed great potential in several control areas. In this concept, the neural network is trained to approximate the action-value function, representing the amount of reward that can be gathered from the very state until the end of the episode. The target values for the training are formulated with the well-known Bellman equation:

$$Q(s_t, a_t; \theta_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t; \theta_t^-). \tag{19}$$

In the equation, $Q(s, a)$ stands for the value of action a in state s. State s' is the next state generated from state s by executing action a. θ_t^- is the weights of the target network, and θ_t is the weights of the online neural network in time step t. DQN is considered as an indirect method since it does not make direct recommendations about action selection. The predicted values of DQN are more like a situation interpretation that reveals the potential of each action in the long run. Consequently, the predicted values have to be exploited to formulate a policy. This way of operation shows that the predicted values of DQN are more like a global heuristic because all the values have an absolute meaning over the entire process.

3.5. Policy-Based

The recent successes of policy-based RL in competitive domains focused tremendous attention to this realm of model-free algorithms. In this concept, the algorithm tunes the neural network to approximate the policy directly in the form of a probability distribution. Consequently, the Policy Gradient (PG) algorithm operates as a dynamic heuristic because it does not present the long term returns of a particular action.

It only reflects the necessities of the given state. Therefore, the predicted values are only comparable to the given branch. As introduced, the output of the PG algorithm is a probability distribution, which is determined by the θ parameters of the neural network. Thus, the optimization task is to tune the weights of the neural network in a way that maximizes the agent's performance indicator function $J(\theta) = J(\pi_\theta)$. This function can be formalized in the episodic set-up as follows,

$$J_{\pi_\theta} = \mathbb{E}\left[\sum_{t=1}^{\tau} \gamma^t r_t\right] \tag{20}$$

Based on Equation (20), it can be seen that the essence of the algorithm is to find the local extreme value in the direction of the largest increase in the discounted cumulated reward. The update rule of the function approximator's θ parameters can be determined based on [39,40], which looks as follows,

$$\theta \leftarrow \theta + \alpha \nabla \log \pi_\theta(s_t, a_t) \sum_{t=1}^{\tau} \gamma^t r_t \tag{21}$$

and operates as follows.

1. First, it initializes the neural network's θ weights and then the training process starts from the first state s_1.
2. The agent and the environment interacts until a terminal state occurs.
3. Calculates the discounted rewards, which are saved in the episode history with the chosen γ discount factor.
4. Then the gradients of the states added to the gradient buffer which will update the neural network's weights if the update frequency encounters.

Where $\pi_\theta(s_t, a_t)$ represents the choice probability assigned to action a in, state s, and α is the learning rate which is one of the most critical hyperparameters of the training process that can be used to determine the extent of change in neural network's weights.

3.6. Planning Agents

The feature that distinguishes RL from other Machine Learning fields that it only requires the model of the control problem and reward scheme to start. Compared to Supervised Learning, this approach greatly simplifies the training process. Moreover, RL algorithms can be applied to control tasks that have not been solved yet, as it does not require a labeled dataset for training. Unfortunately, this concept also has its drawbacks. Notably, RL's trial and error-based nature, which is undoubtedly one of its fascinating virtue, combined with the uncontrolled fashion of how the different episodes follow each other along the training process, makes inevitable the under-representation of experiences. In practice, both value and policy-based RL have tools to mitigate the introduced effect. For DQN, it is the Prioritized Experience Replay, and for the PG algorithm, the gradient buffer has the same purpose. Both tools try to stabilize the training process and prevent the particular algorithm from over-representing specific experiences that locally occurred more frequently than others. Still, this problem can only be solved by infinitely long training, which is not possible, and if it would, we would arrive at the shortcomings of credit assignment. Where immediate rewarding systems behave like heuristics that provide no guarantees, final rewards with discounting concepts assume a particular causality between the consecutive parts of the episodes, which can be untrue. These components make inevitable the deterioration of RL's performance in challenging domains. Nevertheless, the performance of RL algorithms can be enhanced in several ways. Deepmind's researchers introduced AlphaGo Zero [41] that initiates the integration of planning into learning and prediction.

This approach intends to combine the absolute robustness of MCTS with the desired real-time applicability of RL algorithms. Planning integrated into learning utilizes the tree search in every step as an operator that improves and evaluates the policy. Hence it endeavors to improve credit assignment and action selection. In the meantime, planning in the prediction phase utilizes the same combination of the algorithms. It intends to overcome the wrongfully generalized experiences by exploiting the agent's hidden expertise with the tree search's help. Thus it is only used as a policy improvement operator. The reservations against these concepts are related to computational resources. In massive control tasks, both solutions require extensive planning, which can be overwhelming, considering hyperparameter optimization. Consequently, methods that secure real-time applicability and boost performance at once can be incredibly beneficial. This outcome can be realized by reformulating the agent's role as a heuristic inside the tree search selection procedure to make the selection more effective, which enables the algorithm to maintain outstanding performance with considerably less planning time. In these new algorithms, the PG agent is used, since it differentiates the choices more firmly in most states than value-based RL algorithms do, which frequently builds a tree that looks like the result of breadth-first search. This feature is crucial because the tree has to be deep instead of broad, which can only be accomplished by making the most of the appropriated planning time.

3.6.1. Planner-I

The prime question in all tree searches is about the node selection since this procedure can make the search either efficient or lavish. The UCT algorithm's idea is that it is always skeptical about the nodes' value, which firstly assigned than its formulated with backpropagation along with the iterations. Therefore, it systematically overrules the node's value with the desire to try something new because it can not be sure about the value assignment's validity. In the first reformulation of the UCT algorithm, the PG agent's predictions determine all the nodes' values. The concept is that during the training process, the PG agent generally learns how to react in certain cases, but still there will be scenarios where the agent will be in fault thanks to the imperfection of the training concept. Consequently, planning as a tool is used to overrule the wrongfully generalizes experiences. The modified UCT algorithm looks as follows,

$$\overline{p_i} + 2C_p \sqrt{\frac{2 \ln N_i}{n_i}} \tag{22}$$

where the p_i is the predicted choice probability of the considered node. In the introduction of the PG algorithm, it is mentioned that the predicted values are only comparable to the particular branch since they add up to one in every consecutive branch. Therefore, these values are not appropriate to define the value of a state in an absolute manner. These values can sandbag the tree search algorithm through the backpropagation procedure since they are on the same scale. However, this environment has a unique feature regarding that. Notably, one step inside the MCTS algorithm realizes such a small effect on the state vector that the consecutive states' differences are blurred. Hence they can barely be considered Markovian. Moreover, consecutive steps do not represent strategically different scenarios in means like they do in chess or Go, which emphasize the introduced effect. This feature enables the direct use of PG's predicted values without the misleading effect, which initially would require the connection between the consecutive branches like in [42].

3.6.2. Planner-II

In the second case, the value part of the UCT algorithm is the same as in the original, thus identical to the MCTS algorithm. However, the exploration part is different because of PG's prediction rules. Therefore, it also ends up in a neural network driven tree search algorithm but in a different manner. The modified UCT algorithm looks as follows,

$$\overline{X_i} + p_i \sqrt{\frac{2 \ln N_i}{n_i}} \tag{23}$$

This concept creates a different synergic combination between the PG and MCTS since it eliminates the systematic overruling of the exploitation part by exploration, which is now driven by the trained PG agent. Consequently, it initiates a more effective selection procedure, which results in enhanced performance despite the less amount of provided planning time.

3.7. Operation of the Algorithms

For clarification, MCTS can be considered as a model-based search algorithm that operates as follows. In every given state, a search tree is built with a fixed depth, which is determined by the number of executed iterations. After the iterations, the search tree suggests an action according to the utilized action selection policy that seems to be the best over the limited horizon, then the suggested action is executed by the environment, and the process repeats itself in the same way in every step.

The new algorithms operation is very similar to pure MCTS. The difference is that these new algorithms utilize the trained PG agent inside the selection procedure through the reformulation of the UCT-t algorithm. As it makes the selection more effective, it enables the back scaling of the planning time, which results in an algorithm that maintains real-time applicability on an enhanced performance level.

Compared to that, the original RL and SL algorithms do not use any additional techniques; therefore, the action selection is made by the neural network's feedforward function called prediction.

Algorithm 1 presents the actual operation of the proposed Planning agents. There are only two differences between the proposed algorithms. The first is the value assignment, which is a prediction with PG agent for Planner-I and a rollout for Planner-II, and the second is the calculation of the node's UCT value. For further clarification Figure 5 displays the operation of the new algorithms from another aspect.

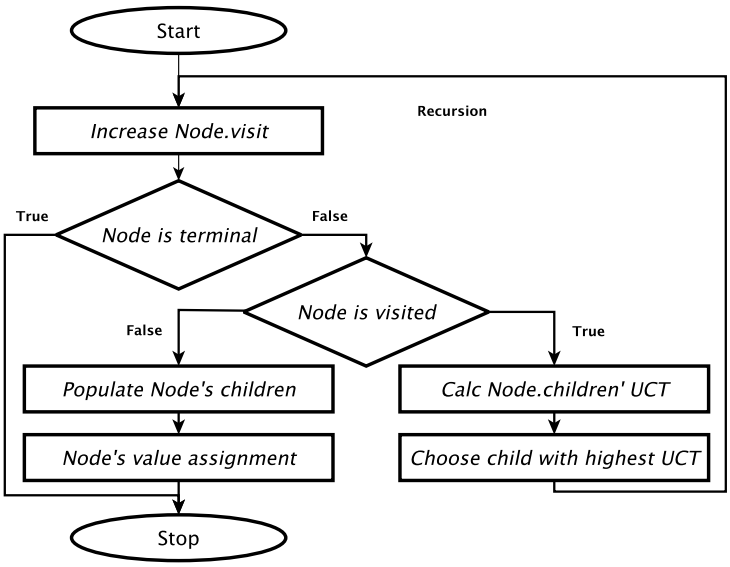


Figure 5. The operation of the planning agents.

Algorithm 1 Planning agent-based control.

```
procedure CONTROL(RootNode, Agent, IterationCnt)
    for i in IterationCnt do
        Iteration(RootNode, Agent)                          ▷ Run modified MCTS
    end for
    Action=argmax(RootNode.Children.Visit)                  ▷ Choose the most visited node
end procedure

procedure ITERATION(Node, Agent)
    Node.Visit += 1
    if Node.Reward == -1 Or Node.Depth == DepthLimit  then  ▷ Terminal node Case
        return Node.Reward
    else if Node.Visit == 0 then                            ▷ Unvisited node Case
        for i in Actions do
            Node.Children[i].Create(Node, Action)
        end for
        if Planner-I then                                   ▷ Predict, if using Planner-I
            Node.Value=Agent.Predict(Node)
        else
            Node.Value=MCTSRollout(Node)                    ▷ Rollout, if using Planner-II
        end if
        return Node.Value
    else                                                    ▷ Visited node Case
        for child in Node.Children do
            child.UCT();
        end for
        BestChild=Node.Children(argmax(Node.Children.UCT))
        v=Iteration(BestChild, Agent)                       ▷ Recursive Call
        Node.Value+=v
        return v
    end if
end procedure
```

4. Results

In describing the agent's state vector, numerous claims are made about the lookahead part's role and necessity. Consequently, the lookahead's legitimacy has to be justified, which is realized by comparing two types of agents' performance on tens of thousands of randomly generated trajectories, while the environmental seeds are fixed to ensure representativity. In this testing scenario, each agent takes the same number of steps in the trajectory by starting from the exact point. The previously introduced reward scheme evaluates each step, and this method goes for all courses. Table 3 shows the average cumulated reward and the share in failed episodes for all two agents types.

Table 3. Statistic performance of algorithms with and without lookahead sensor information.

	DQN	PG	PG without Lookahead
Average cumulated reward [-]	407	420	381
Share in failed episodes [%]	3.2	3.43	18.14

The percentage of the agent which does not use lookahead is approximately six times higher than the one that does, and obviously the average cumulated reward reflects the same. To understand what is lacking, the assessment has to go beyond statistics. Figure 6 presents different scenarios to give an intuitive sense of the two types of agents' operation and their differences. The agent that does not use the full sensor information operates as an improperly tuned controller that overshoots. Therefore, it can not prepare for a sharper turn in the trajectory because it misses the right time to act. Consequently, it significantly

deviates from the centerline in sharp turns. Moreover, the agent can not hold the vehicle in the lane in many cases. In the meantime, the agents with lookahead deliberately start to deviate from the centerline to hold the vehicle in the lane, which means that they learned how to react if the trajectory ahead of the vehicle evolves in a certain way. In some sense, this behavior can be interpreted as seeing beyond a local optimum and choosing a global one instead.

The tree search algorithm's operation time is introduced as a major concern since a tree search has to be carried out in every step to choose from the possible actions. Therefore, it is crucial to find the right trade-off between the appropriated planning time and performance for the pure MCTS algorithm because the proper amount of lookahead is a necessity when sharp turns or turn combinations are ahead of the vehicle. In the original set-up, the MCTS algorithm uses the same 0.1 s time step as the environment. In such control tasks, a 5 s lookahead can be considered as a minimum for accurately controlling the vehicle. Consequently, a 50-layer deep tree has to be built in every step to reach the required 5 s lookahead. Unfortunately, this amount of planning can not be carried out in every step because the computational resources are lacking. Thus, a different strategy is formulated where the MCTS's time step is changed to 0.5 s. This approach significantly reduces the required planning time by reaching the desired depth in the tenth layer. It also gives an intuition that the direction of the next action matters more than the exact steering angle because the state-space created by the MCTS differs from the environment's state-space ahead of the vehicle. Still, it controls the vehicle in the right fashion. This concept enables to reduce the number of populated nodes, while the algorithm can act on time and take sharp turns and turn combination. Table 4 shows the average cumulated reward and a share in failed episodes for the MCTS with 0.1 s and 0.5 s step sizes where both algorithms get the same planning time and the same courses to drive on.

Table 4. Statistic performance of Monte Carlo tree search (MCTS) with 0.1 s step size and MCTS-I with 0.5 s step size.

	MCTS-I	**MCTS**
Average cumulated reward [-]	301	442
Share in failed episodes [%]	36.4	3.8

The difference between the average scores is tremendous, just as the gap between the share in failed episodes, where the MCTS-I algorithm's percentage is almost ten times higher than the modified MCTS's. These results suggest that this new approach is beneficial and enables using such an algorithm on a high-performance level.

In the training of a neural network with SL, the most critical step is data preparation. First, all the x input vectors have to be normalized, which means it has to be done in all training samples. It is hugely influential since it helps avoid numerical problems such as vanishing and exploding gradients and makes the optimization problem better conditioned, which means less tuning can be enough. The representation of all classes equally is also a key component in reaching the highest possible accuracy. It is also essential to filter the training samples because there are episodes where the MCTS fails to hold the vehicle in the lane. Hence theses experiences are omitted. Finally, the training dataset also has to be shuffled to maintain the classes' equal representation in every batch. The hyperparameters such as the learning rate, batch size, and number of epochs are chosen with trial and error.

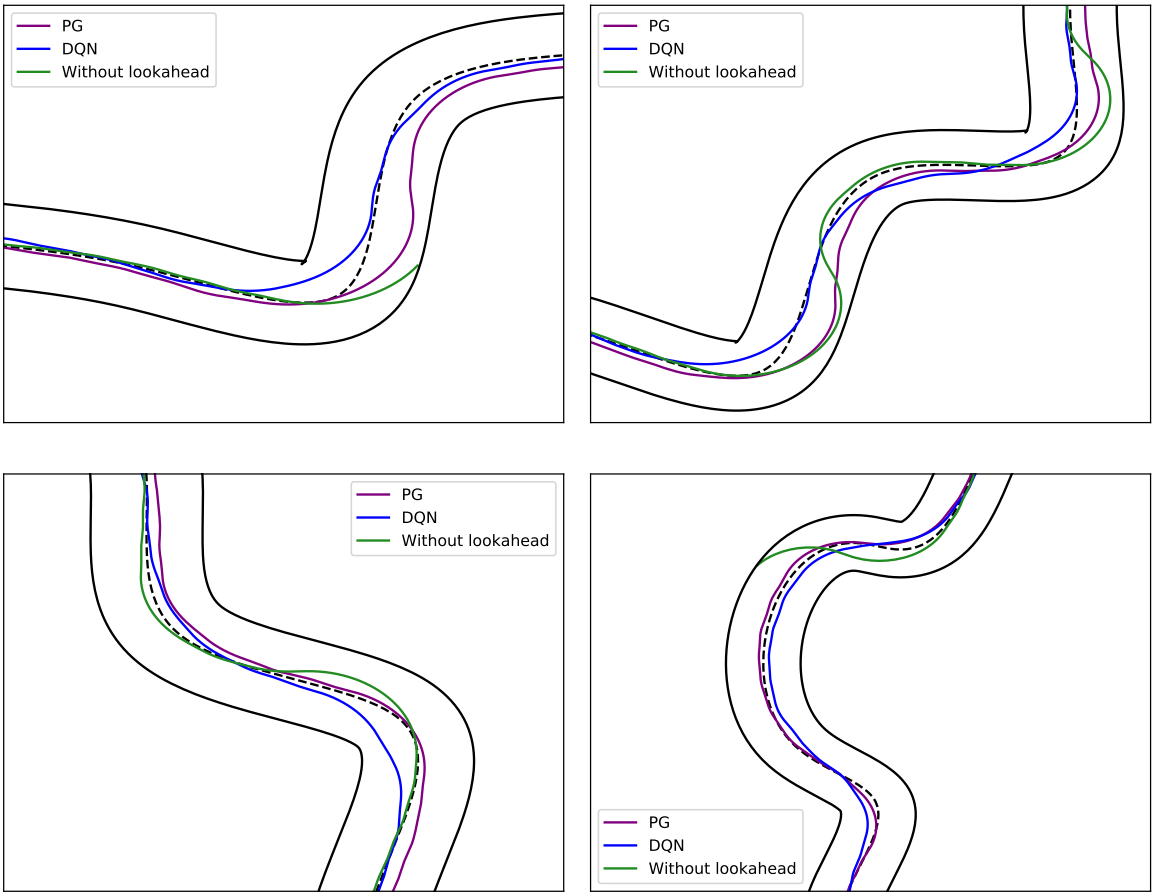


Figure 6. Comparison of Agents with and without lookahead information.

4.1. Statistical Comparison

Before starting to compare the performances, the convergence of both RL algorithms has to be evaluated. Figure 7 shows the convergence of the PG and DQN algorithms. The PG converges faster and a more stable manner than the DQN algorithm, and it also reaches a higher average score. Therefore, the PG algorithm seems superior by any measure. Consequently, the same relation is expected between the performances.

In the statistical comparisons, all the algorithms have to take five hundred steps starting from the same position on the exact ten thousand episodes, which are ensured with environment seeds to hold up representativity. Figure 8a shows the performance of both the MCTS and SL algorithms through the frequency of cumulated rewards collected into ten-point wide intervals. MCTS scores at least 450 points in 86% of the episodes, which means its performance is 90% or higher, while it is only true the 70% of the SL's episodes. In the meantime, the SL algorithm has a considerably higher share in failed episodes than the MCTS algorithm has. Still, looking at the overall performance and the similarities in the characteristics suggests that the training of the SL agent is successful. It is interesting, though, that the MCTS algorithm can not solve all the episodes. This phenomenon is caused by the trade-off between performance and planning time. The MCTS's share in failed episodes can be decreased to zero, but it would cost unreasonably much planning that would seriously question the utilization of such an algorithm for this particular control task.

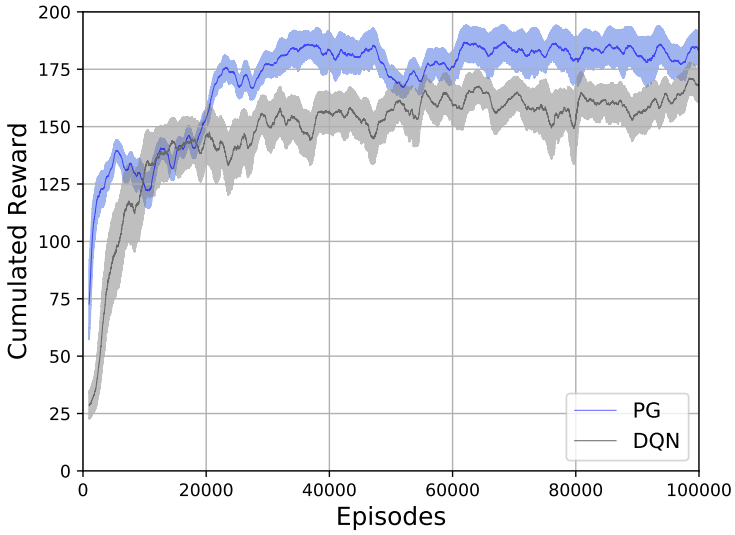
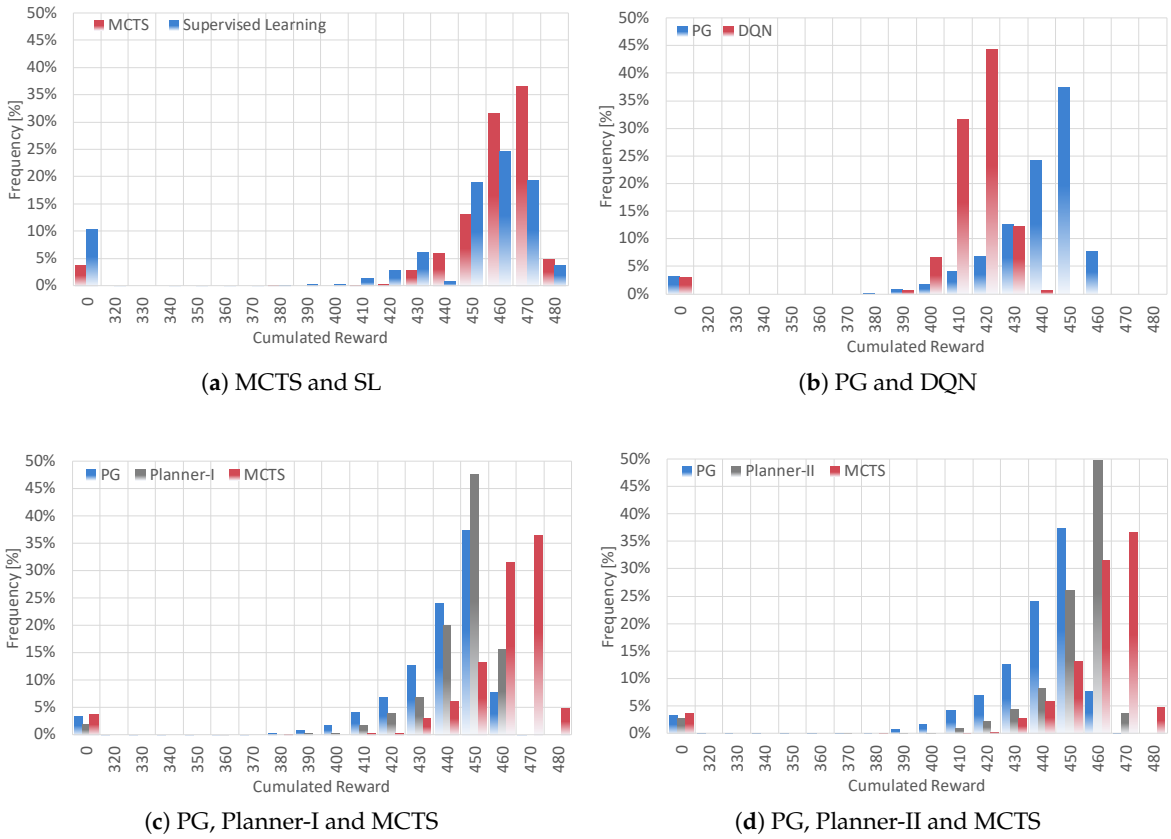


Figure 7. Convergence of PG and DQN algorithms.

The statistical comparison of the PG and DQN algorithms in Figure 8b confirms the expectations. The PG algorithm reaches a higher performance level as it has a greater share in the best possible scores then the DQN does. However, the DQN share in failed episodes in slightly lower than the PG's. Therefore, it performs better in some sense. By comparing to Figure 8a, RL-based solutions can not reach the performance level of the MCTS or SL, but in terms of failed episodes, the RL-based solutions perform better. The relation between the RL's and MCTS's performance yields that the combined algorithm can synergize the advantages to mitigate the weaknesses to overperform both its ancestors.

Figure 8c,d presents the performance of the two new algorithms, which are the combination of RL and MCTS. The Planner-I algorithm's performance is shown in Figure 8c, where it is compared to its ancestors. The Planner-I algorithm boosts the performance of the PG algorithm and pushes it closer to the performance level of MCTS. In the meantime, it decreases the share in failed episodes nearly to the half of the MCTS's share, while it is also lower than the PG's share. Compared to the SL, it performs above 90% in only 60% of the episodes, which is slightly lower, but SL's share in failed episodes is five times higher. Therefore, the overall performance is better than the SL agent. The Planner-II algorithm realizes the same characteristics, but it comes closer to the performance of MCTS than Planner-I does. Compared to MCTS's 86%, it scores above 90% in 79% of the episodes, and its share in failed episodes is less then MCTS's, but it is higher than the share of Planner-I. Compared to SL, the Planner-II's performance is better in the region above 90%, and its share in failed episodes is more than three times lower.

For further comparison, all the algorithms average cumulated reward and share in failed episodes are presented in Table 5, where MCTS-II refers to the pure MCTS with the same planning time that is appropriated for the Planner-I and Planner-II algorithms.



(**a**) MCTS and SL

(**b**) PG and DQN

(**c**) PG, Planner-I and MCTS

(**d**) PG, Planner-II and MCTS

Figure 8. Comparison of the algorithms' performance on the long run.

Table 5. Statistic performance of all the algorithms.

	MCTS	MCTS-II	MCTS-I	PG	DQN	PG-II	Planner-I	Planner-II
Average score [-]	442	395	301	420	407	380	435	437
Failed episodes [%]	3.8	15.1	36.4	3.43	3.2	18.14	2	3

The MCTS-I that does not utilize the introduced concept about the step sizes has the absolute worst performance by all measures. The MCTS-I is followed by the PG-II, which does not utilize the lookahead sensor information part as other agent does. This part of the ranking justifies the paper's side contributions about the state vector for RL and the modifications in MCTS since the algorithms that do not utilize the mentioned concepts are got stuck on a mediocre performance level. The next algorithm in line is the MCTS-II, which does not utilize RL in the node selection procedure. The performance of MCTS-II highlights the importance of the integration of RL into MCTS's node selection procedure as this is the only difference compared to the new algorithms. In the meantime, the average score of MCTS-II is not even close to the scores of the new algorithms, not to mention the share in failed episodes, which is seven times more compared to Planner-I and five times to Planner-II. The average score of pure ML-based methods such as the DQN, PG, and SL algorithms is close to each other, but the RL based methods have a lower share in failed episodes. Therefore, overall, RL outperforms SL. Both of the new algorithms enhance PG's performance, which can be observed by comparing the average scores, and decreases the number of failed episodes. Compared to MCTS, both of the new algorithms decrease the number of failed episodes, and they come close to the average score of MCTS, but they can not reach it. However, both new algorithms

have a feature that exceeds MCTS, which is real-time applicability. In conclusion, the synergy between RL and MCTS has resulted in algorithms that enhance RL's performance and hone MCTS in terms of failed episodes, while they maintained real-time applicability.

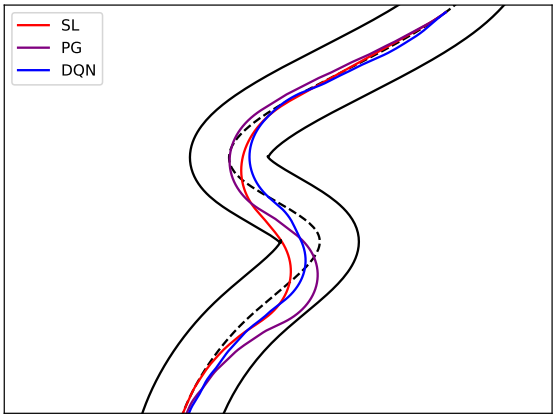
4.2. Strategic Comparison

Statistical comparison is an excellent way of presenting results, but unfortunately it cannot put the means behind the numbers. Thus, to do so, a strategic comparison is carried out, which reveals the reasons that formulate the statistical results. Figure 9 presents one complicated and one simple section of a trajectory. The planning and training based controllers are separated for clarity. The comparison of the agents for the complicated situation is displayed in Figure 9a. In this figure, the PG algorithm carries out an exciting behavior before taking turns. Notably, it slightly deviates from the centerline to prepare for the turn, to hold the vehicle on the centerline during the turn. This behavior seems wise, but in turn combinations it stays on the outer path and misses the right time to act, which results in a significant overshoot at the exit of the turn combination. Consequently, it finds an excellent strategy for entering into a turn, but not for exiting it. In the meantime, DQN and SL choose similar strategies both for entering and exiting the turns. They cut the turns from the inner path. The difference is that the DQN agent cut the corner more firmly than SL does, and it tries to go back to the centerline immediately. The reason for this is in the utilization of the lookahead. The SL agent is trained with the dataset generated with the MCTS that considers the entire lookahead in every step, while the DQN algorithm gets a hundred thousand episodes to figure out how to use it. Consequently, the SL agent behaves like considering the whole turn combination, not just the next turn. However, at the end of the turn, the SL still misses the right time to act, which results in the same deviation as the DQN does. This behavior can be interpreted as a flaw in the generalization of the experiences during the training process. At the bottom of Figure 9a, it can be seen that the SL agent quickly goes back to the centerline after the deviation, while DQN seems to stay deviated, and PG endeavors to go back but more slowly. This behavior is observed several times; therefore, it may be why the DQN reaches the lowest average cumulated reward from all the trained ML-based agents.

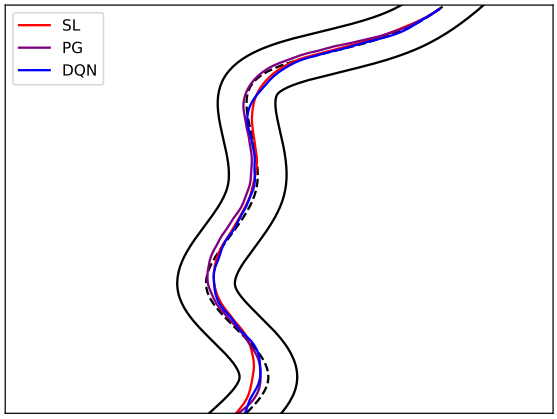
Figure 9c presents the strategic comparison of all the algorithms that use planning. MCTS-II is the same as the pure MCTS, but the provided planning time is identical to the new algorithms. Its trajectory shows that the made decisions lack the wisdom of a deep tree that covers the turn combination. Consequently, it cannot hold the vehicle in the lane. The new algorithms' style and strategy show the PG algorithm's marks, mostly in the entering of the turn. In the meantime, planning's impact is also remarkable, as the overshoot of the new algorithms is considerably lower than the pure PG's. It is interesting, though, that Planner-I has a lower overshoot than Planner-II since it uses the PG to determine the nodes' value, which is definitely a more substantial impact on the UCT algorithms final value than the regularization of the explore component. This phenomenon shows the effect of the backpropagation on changing the firstly assigned values of the nodes. The PG algorithm naturally has a bigger overshoot. Consequently, Planner-I reaches the edge of the lane in earlier steps during the search, and it finds out the flaws of the PG's strategy, so it has more steps to interfere. In the meantime, the UCT algorithm of Planner-II utilizes a different synergy between RL and MCTS, where RL has a lower impact, which in this particular case, prevents the algorithm from finding the flaws in PG's strategy on time. The MCTS algorithm's trajectory shows that the depth of its tree is proper since both the entering and the exiting of the turn combination are precise. Despite the end of the turn combination, the trajectory of the MCTS shows the same style marks as the SL does, which reassures the previously introduced ideas about the SL's strategy.

Figure 9b,d presents a much simpler trajectory section to shows how the algorithms behave in ordinary scenarios. In Figure 9b, the ML-based agents are compared. All the agents have the same

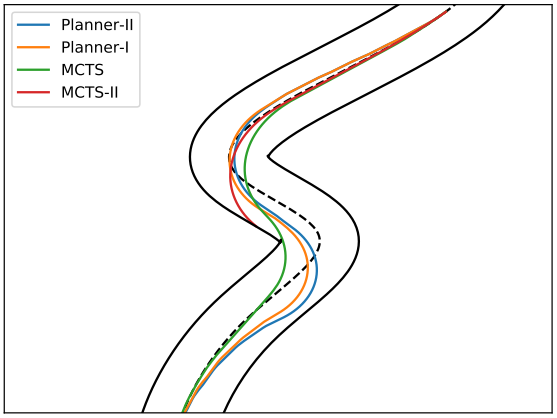
behavior as in the more complex case presented in Figure 9a, but the strategical differences seem less significant. Still, PG uses the outer path at the beginning of the turn to stay on the centerline. Since the turns are less sharp, the deviation at the end of the turn is marginal. The same goes for the strategy of the DQN and SL agents. They cut the turns from the inner path. However, in this particular case, the DQN agent cuts the turns more accurately than the SL. The planning based algorithms for the simpler trajectory section are presented in Figure 9d. The effect of the PG on the new algorithms is still recognizable, but as in the previous case, the deviations from the centerline are less significant, and in this section, the behavior of Planner-I and Planner-II are very similar. Therefore, the effect of the differences in the UCT algorithm can mostly be observed in sharper turns. In the meantime, MCTS holds the vehicle precisely on the centerline between turns and deviates from it as less as possible and only if it is necessary.



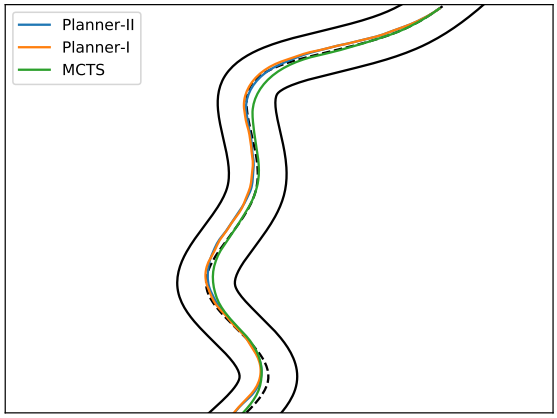
(**a**) SL, PG, and DQN on sharp turn combination



(**b**) SL, PG, and DQN on easy path



(**c**) Search based solutions on sharp turn combination

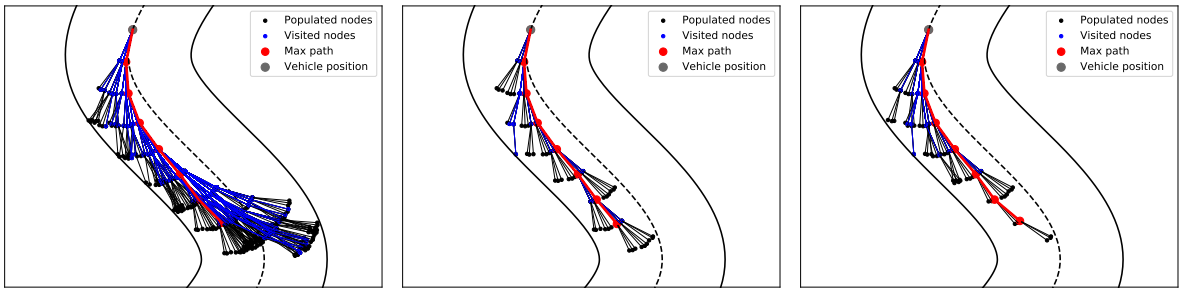


(**d**) Search based solutions on easy path

Figure 9. Strategical comparison of the algorithms on various trajectory sections.

Along with the presentation of the new algorithms, it is mentioned that the provided planning time is remarkably decreased, and still by the integration of RL into the UCT algorithm, the selection procedure becomes more efficient, which results in enhanced performance. However, it is hard to interpret what happens inside the tree and what efficient node selection means. Figure 10 resolves this issue and supports

the interpretation of used phrases by visualizing the tree built for a single decision. Figure 10a–c are created in a sharp turn combination to see how the tree covers the lane ahead of the vehicle, where all three algorithms starts the planning from the same position. All the subfigures have the max-path marked with red color, which only reflects the suggestion of the currently carried out search where only the action of the first step is realized since the tree search suggest one action at the time. Moreover, the vehicle's next position is not identical to the visualized node because the step size of the environment and the MCTS is different, as mentioned previously. Consequently, the real trajectory can and will deviate from the max-path. However, the MCTS's max-path style has the highest possibility to become a reality, as it is honed with the greatest amount of planning compared to the new algorithms. Figure 10a displays the search tree built by the pure MCTS algorithm. The nodes with blue colors are the ones that the tree search algorithm visited during the iterative planning process and the black ones that the algorithm populated. The difference between the number of blue and black nodes shows the strength of the MCTS algorithm, as it does not need to visit all the nodes to provide such great performance. By comparing Figure 10b,c with Figure 10a, the effectiveness of the new algorithm's selection procedure can be seen through the depth of the different trees as the new algorithms can use nearly the same length of road section ahead of the vehicle as MCTS uses in every decision. These figures also shows the paper's main contribution through the difference between the number of populated nodes. This visualization gives a great intuition on how to interpret the effect of decreased planning time and highlights that despite the narrower tree, the utilization of trained agents as heuristics prevents the algorithms from making bad moves. Moreover, the new algorithms outperform pure MCTS in terms of failed episodes and real-time applicability.



(**a**) MCTS (**b**) Planner-I (**c**) Planner-II

Figure 10. Search trees built by different algortihms.

5. Conclusions

This paper presents two new algorithms for the lateral vehicle control problem of a dynamic nonlinear single track vehicle model in the form of lane-keeping, along with the in-depth statistical and strategic comparison of six different ML and search-based algorithms. Two more side contributions are introduced: one in the implementation of the MCTS algorithm and one in the formulation of the RL-agents state vector. In the justification of these side contributions, the comparisons showed that both the modification of MCTS and the state vector complied with lookahead information is necessary to utilize such sophisticated algorithms on a high-performance level in massive control tasks. The new algorithms reformulate the UCT algorithm to make the tree search's node selection procedure more effective by utilizing RL as a heuristic. The new synergy of RL and MCTS enhances RL's performance and decreases the failed episodes, belove the share of both MCTS and PG. Moreover, every other presented algorithms' average performance falls short of the performance of the new algorithms. The only measure where the new algorithms fail to exceed MCTS is the average score, but they come close. However, the new algorithms can maintain

real-time applicability, which seems a fair trade-off considering the slightly lower average performance. Consequently, new algorithms overcome the overall performance of their ancestors. It is essential to mention that no domain-specific cut-offs are utilized in the tree searches, which show that the introduced new algorithms may be applied in a wild variety of control problems as its strength does not depend on handcrafted features or domain-specific knowledge.

Author Contributions: Conceptualization, B.K. and T.B.; methodology, B.K.; software, B.K. and F.H.; validation, T.B.; formal analysis, F.H.; investigation, B.K. and T.B.; writing—original draft preparation, B.K., T.B., and F.H.; visualization, B.K. and F.H.; supervision, T.B. All authors have read and agreed to the published version of the manuscript.

Funding: The research has been supported by the National Research, Development and Innovation Office (NKFIH) through the project "National Lab for Autonomous Systems" (NKFIH-869/2020). The research was also supported by the Hungarian Government and co-financed by the European Social Fund through the project "Talent management in autonomous vehicle control technologies" (EFOP-3.6.3-VEKOP-16-2017-00001).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript.

ALVINN	Autonomous Land Vehicle in a Neural Network
C-LSTM	Convolution Long-Short-Term-Memory Neural Network
CNN	Convolution Neural Networks
DDAC	Deep Deterministic Actor-Critic
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DQN	Deep Q Networks
IRL	Inverse Reinforcement Learning
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
PG	Policy Gradient
RL	Reinforcement Learning
SL	Supervised Learning
UCT	Upper Confidence bound applied for Trees
VAE	Variational AutoEncoder

References

1. Deng, L.; Yang, M.; Qian, Y.; Wang, C.; Wang, B. CNN based semantic segmentation for urban traffic scenes using fisheye camera. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 231–236.
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*; Curran Associates, Inc.: New York, NY, USA, 2012; pp. 1097–1105.
3. Aradi, S. Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2020**, 1–20. [CrossRef]
4. Kuutti, S.; Bowden, R.; Jin, Y.; Barber, P.; Fallah, S. A Survey of Deep Learning Applications to Autonomous Vehicle Control. *IEEE Trans. Intell. Transp. Syst.* **2020**. [CrossRef]
5. Pomerleau, D.A. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*; Morgan-Kaufmann: Burlington, MA, USA, 1989; pp. 305–313.

6. Pomerleau, D.A. Neural network vision for robot driving. In *The Handbook of Brain Theory and Neural Networks*; Citeseer: Princeton, NJ, USA, 1996.
7. Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to end learning for self-driving cars. *arXiv* **2016**, arXiv:1604.07316.
8. Muller, U.; Ben, J.; Cosatto, E.; Flepp, B.; Cun, Y.L. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2006; pp. 739–746.
9. Rausch, V.; Hansen, A.; Solowjow, E.; Liu, C.; Kreuzer, E.; Hedrick, J.K. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In Proceedings of the 2017 American Control Conference (ACC), Seattle, WA, USA, 24–26 May 2017; pp. 4914–4919.
10. Mechanical Simulation Corporation. CarSim. Available online: https://www.carsim.com (accessed on 8 August 2020).
11. Eraqi, H.M.; Moustafa, M.N.; Honer, J. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *arXiv* **2017**, arXiv:1710.03804.
12. Rothe, R.; Timofte, R.; Van Gool, L. Dex: Deep expectation of apparent age from a single image. In Proceedings of the IEEE international conference on computer vision workshops, Santiago, Chile, 7–13 December 2015; pp. 10–15.
13. Yu, G.; Sethi, I.K. Road-following with continuous learning. In Proceedings of the Intelligent Vehicles' 95. Symposium, Detroit, MI, USA, 25–26 September 1995; pp. 412–417.
14. Wang, P.; Chan, C.Y.; de La Fortelle, A. A reinforcement learning based approach for automated lane change maneuvers. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1379–1384.
15. Dai, X.; Li, C.K.; Rad, A.B. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. *IEEE Trans. Intell. Transp. Syst.* **2005**, *6*, 285–293. [CrossRef]
16. Desjardins, C.; Chaib-Draa, B. Cooperative adaptive cruise control: A reinforcement learning approach. *IEEE Trans. Intell. Transp. Syst.* **2011**, *12*, 1248–1260. [CrossRef]
17. Huang, Z.; Xu, X.; He, H.; Tan, J.; Sun, Z. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. *IEEE Trans. Syst. Man, Cybern. Syst.* **2017**, *49*, 730–741. [CrossRef]
18. Chae, H.; Kang, C.M.; Kim, B.; Kim, J.; Chung, C.C.; Choi, J.W. Autonomous braking system via deep reinforcement learning. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 1–6.
19. Zhao, D.; Wang, B.; Liu, D. A supervised actor–critic approach for adaptive cruise control. *Soft Comput.* **2013**, *17*, 2089–2099. [CrossRef]
20. Wang, B.; Zhao, D.; Li, C.; Dai, Y. Design and implementation of an adaptive cruise control system based on supervised actor-critic learning. In Proceedings of the 2015 5th International Conference on Information Science and Technology (ICIST), Changsha, China, 24–26 April 2015; pp. 243–248.
21. Hegedűs, T.; Németh, B.; Gáspár, P. Challenges and Possibilities of Overtaking Strategies for Autonomous Vehicles. *Period. Polytech. Transp. Eng.* **2020**, *48*, 320–326. [CrossRef]
22. Xia, W.; Li, H.; Li, B. A control strategy of autonomous vehicles based on deep reinforcement learning. In Proceedings of the 2016 9th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 10–11 December 2016; Volume 2, pp. 198–201.
23. Sallab, A.E.; Abdou, M.; Perot, E.; Yogamani, S. End-to-end deep reinforcement learning for lane keeping assist. *arXiv* **2016**, arXiv:1612.04340.
24. Qian, L.; Xu, X.; Zeng, Y.; Huang, J. Deep, Consistent Behavioral Decision Making with Planning Features for Autonomous Vehicles. *Electronics* **2019**, *8*, 1492. [CrossRef]
25. Porav, H.; Newman, P. Imminent collision mitigation with reinforcement learning and vision. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 958–964.
26. Kuderer, M.; Gulati, S.; Burgard, W. Learning driving styles for autonomous vehicles from demonstration. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 2641–2646.

27. Ly, A.O.; Akhloufi, M.A. Learning to drive by imitation: An overview of deep behavior cloning methods. *IEEE Trans. Intell. Veh.* **2020**. [CrossRef]
28. Lin, Y.; McPhee, J.; Azad, N.L. Longitudinal dynamic versus kinematic models for car-following control using deep reinforcement learning. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 1504–1510.
29. Fehér, Á.; Aradi, S.; Hegedüs, F.; Bécsi, T.; Gáspár, P. Hybrid DDPG Approach for Vehicle Motion Planning. In Proceedings of the ICINCO 2019—16th International Conference on Informatics in Control, Automation and Robotics, Prague, Czech Republic, 29–31 July 2019.
30. Pacejka, H. *Tire and Vehicle Dynamics*; Elsevier: Amsterdam, The Netherlands, 2005.
31. Hegedüs, F.; Bécsi, T.; Aradi, S.; Gáspár, P. Motion Planning for Highly Automated Road Vehicles with a Hybrid Approach Using Nonlinear Optimization and Artificial Neural Networks. *Stroj. Vestn. J. Mech. Eng.* **2019**, *65*, 148–160. [CrossRef]
32. Thompson, W.R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **1933**, *25*, 285–294. [CrossRef]
33. Bush, R.R.; Mosteller, F. A stochastic model with applications to learning. *Ann. Math. Stat.* **1953**, *24*, 559–585. [CrossRef]
34. Lattimore, T.; Szepesvári, C. *Bandit Algorithms*; Cambridge University Press: Cambridge, UK, 2018; in press.
35. Chaslot, G.M.B.; Winands, M.H.; van Den Herik, H.J. Parallel monte-carlo tree search. In *International Conference on Computers and Games*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 60–71.
36. Kocsis, L.; Szepesvári, C. Bandit based monte-carlo planning. In *European Conference on Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 282–293.
37. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. Games* **2012**, *4*, 1–43. [CrossRef]
38. Sutton, R.S.; Barto, A.G. *Introduction to Reinforcement Learning*; MIT Press: Cambridge, UK, 1998; Volume 135.
39. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2000; pp. 1057–1063.
40. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
41. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of go without human knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef] [PubMed]
42. Bécsi, T.; Szabó, Á.; Kővári, B.; Aradi, S.; Gáspár, P. Reinforcement Learning Based Control Design for a Floating Piston Pneumatic Gearbox Actuator. *IEEE Access* **2020**, *8*, 147295–147312. [CrossRef]