Path Following with Deep Reinforcement Learning for Autonomous Cars

Khaled Alomari[a], Ricardo Carrillo Mendoza, Daniel Goehring and Raúl Rojas
Dahlem Center for Machine Learning and Robotics - Freie Universität Berlin, Arnimallee 7, 14195 Berlin, Germany

Keywords: Deep Reinforcement Learning, Deep Deterministic Policy Gradient, Path-following, Advanced Driver Assistance Systems, Autonomous Vehicles.

Abstract: Path-following for autonomous vehicles is a challenging task. Choosing the appropriate controller to apply typical linear/nonlinear control theory methods demands intensive investigation on the dynamics and kinematics of the system. Furthermore, the non-linearity of the system's dynamics, the complication of its analytical description, disturbances, and the influence of sensor noise, raise the need for adaptive control methods for reaching optimal performance. In the context of this paper, a Deep Reinforcement Learning (DRL) approach with Deep Deterministic Policy Gradient (DDPG) is employed for path tracking of an autonomous model vehicle. The RL agent is trained in a 3D simulation environment. It interacts with the unknown environment and accumulates experiences to update the Deep Neural Network. The algorithm learns a policy (sequence of control actions) that solves the designed optimization objective. The agent is trained to calculate heading angles to follow a path with minimal cross-track error. In the final evaluation, to prove the trained policy's dynamic, we analyzed the learned steering policy strength to respond to more extensive and smaller steering values with keeping the cross-track error as small as possible. In conclusion, the agent could drive around the track for several loops without exceeding the maximum tolerated deviation, moreover, with reasonable orientation error.

1 INTRODUCTION

Autonomous driving has gained exceptional attention as an essential research topic in recent years. It is forming the future of transportation. This tendency is the product of the increased efforts from several automotive manufacturers to integrate more Advanced Driver Assistance Systems (ADAS) with various automation features in their modern cars. Moreover, they endeavor to test them on public roads around the globe to test their stability under environmental uncertainties (Chan, 2017). To develop fully autonomous cars, High-performance assistance systems are needed; developing them demands intensive investigation on the dynamics and kinematics states of the system under a wide range of driving conditions in complex environments (Martinsen and Lekkas, 2018).

Choosing the appropriate controller, applying typical linear/nonlinear control theory methods, is not always possible due to the non-linearity of the dynamics, sensor noise influence, disturbances, and unknown parameters. Thus, artificial intelligence approaches to design an adaptive controller that can reach optimal performance were raised. Reinforcement learning is a framework by which a control policy can be found for a system with unknown dynamics (Hall et al., 2011) (Calzolari et al., 2017).

[a] https://orcid.org/0000-0001-7248-0056

2 RELATED WORK

Reinforcement Learning (RL) is a category of machine learning in which an agent learns from interacting with an environment (i.e., the agent accumulates the perception about the environment from experimental trials and simple relative feedback received). The goal is to let the agent learn the best possible actions in an environment to attain its goals efficiently. During the process, the agent is capable of adapting to the environment to maximize future rewards actively (Sutton and Barto, 2018)(Kaelbling et al., 1996).

In (Wang et al., 2018), TORCS environment was used to train an RL-agent. Their goal was to drive the car at high speed in the center of the road without crashing other cars. A set of sensor data, including track points, car speed, orientation, and the deviation between the vehicle's longitudinal axis and the ideal

Alomari, K., Mendoza, R., Goehring, D. and Rojas, R.
Path Following with Deep Reinforcement Learning for Autonomous Cars.
DOI: 10.5220/0010715400003061
In *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems (ROBOVIS 2021)*, pages 173-181
ISBN: 978-989-758-537-1
Copyright © 2021 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

track, are used as inputs for an adopted DDPG algorithm. Decisions are made for either no throttle or full throttle, no braking or full braking, or full steering to the left or the right.

In contrast to the previous approach, Jaritz et al. (Jaritz et al., 2018) proposed to learn the end-to-end driving control of an autonomous car using only images from a forward-facing monocular camera. They trained the agent in a simulated environment to learn full control -steering, brake, gas, and even hand brake to a pressure drifting- using the A3C algorithm. Similarly, in (Kendall et al., 2018), they used the monocular images to train an agent to follow a lane. In (Li et al., 2019), a perception module in the form of a Convolutional Neural Network (CNN) is utilized to extract track information from the image data of a racing simulator. This information is then combined with information about the driving condition, such as vehicle position, speed, and orientation, to train an RL agent using DPG to predict a continuous steering angle.

Reinforcement Learning methods are incredibly time-consuming, and millions of experiences need to be accumulated to learn complicated tasks accurately. As a consequence, most of the successfully robotic RL agents are trained in a simulation environment. As a result, the training process can be automated efficiently, and the dangerous circumstances caused by trial-and-error are reserved from the real world. Still, The RL agent should be tested in a real situation.

In this paper, the state-of-the-art DRL algorithm Deep Deterministic Policy Gradient is employed for path tracking control of an autonomous model vehicle. The RL agent is trained in a 3D simulation environment. It interacts with the anonymous environment and accumulates experiences to update the Deep Neural Network. The agent is trained to generate and execute heading angles to follow a path with a minimum cross-track error. The deviation between the target and actual path is mainly considered, which could be significantly reduced. The benefit of our implementation is that it brings close the implementation from simulation to real-world application. Moreover, the simulator engine used has dynamic parameters which make the training closer to reality. It also has other sensors used in autonomous driving, such as LIDAR, stereo camera, and odometry.

3 METHODS AND SETUP

This section describes in detail the methods used to implement the environment and the reinforcement learning agent.

3.1 Environment Setup

This subsection describes the setting up of an environment to apply and test RL-agent. The built environment integrates the AutoMiny-simulator (Schmidt et al., 2019) with the open-source library OpenAI Gym (Brockman et al., 2016). In the following, the environment structure will be described in detail, along with the AutoMiny-simulator.

3.1.1 AutoMiny Simulator

The AutoMiny-Simulator is integrated with ROS and is based on the robust physics engine Gazebo. It creates a comprehensive 3D dynamic robot environment capable of recreating the physical driving setup of the agent. Furthermore, it emulates dynamic parameters such as static and dynamic friction and aerodynamics. It considers the vehicle dynamics parameters like wheel acceleration, wheel material, and torque, making it more reliable to transfer the developed algorithms into the model car and test them.

The simulator is developed based on the existing AutoMiny project at the Freie University of Berlin (Alomari et al., 2020). It provides all nodes and topics used by the physical AutoMiny car (Schmidt et al., 2019). The Kinematic model of the car has been parameterized using its URDF description. All the onboard sensors associated with AutoMiny, like the RGBD camera, IMU, and the laser scanner, are provided in the simulated car.

3.1.2 GazeboAutominyEnv

The developed environment, named "GazeboAutominyEnv," links the AutoMiny-simulator with the open-source library OpenAI Gym. Figure 1 describes the structure of the developed environment. First, the simulator is launched while initializing the environment class. Then, at the rate of 30 Hz, the observation function inquires raw observation data from the simulator by subscribing to pre-defined suitable topics. Moreover, it calls the path points. Finally, the observation function collects data and returns them normalized in a box form (continuous values).

At each iteration, the step function requests the last observation data. This supplied information defines a target point on the path and formalizes a time-related state. The state-space is fed to both the actor and the critic networks. The actor-network processes this data and predicts an action based on the policy. The estimated action is sent back to the environment and supplied to the critic network as well. The step function controls the action and publishes it to the

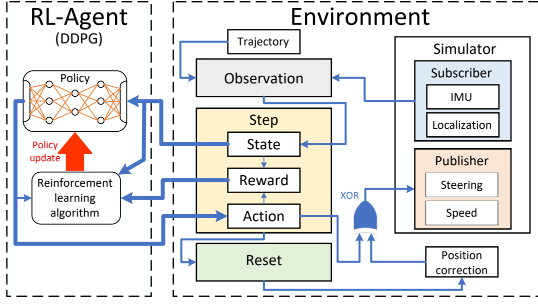


Figure 1: GazeboAutominyEnv Environment Setup Structure; At each iteration, the step function fed the state parameters to the RL-agent, which predicts an action based on the policy. The estimated action is sent back to the step function to process it. After taking action, the reward function evaluates the new error value and either reward the action or penalize it.

simulator topics. After taking action, the step function requests new observation data. This data is used to update the state and calculate the error between the estimated and actual controlled value. Meantime, it evaluates this error value and either reward the action or penalize it. The reward value is delivered to the critic network. The critic network then updates the policy. However, the process on the agent side will be described more in detail in Section 3.2.

Once the episode is completed or terminated, the step function calls the reset function, which moves the agent back to an initial position (either fixed or arbitrary) using a navigation system based on a vector virtual force field (Alomari et al., ress).

3.2 Design of the RL-agent

For this work, the Deep Deterministic Policy Gradient (Lillicrap et al., 2016) is employed. It is one of the straightforward algorithms in Deep Reinforcement Learning, which is suitable for continuous and discrete action spaces. Furthermore, DDPG has an actor-critic architecture and concurrently learns a policy and a Q-function. The agent implementation is utilized by Keras-rl. Keras-rl operates with the OpenAI Gym environment. The deep learning part is accomplished in the Keras framework with TensorFlow backend and allows us to establish custom actor-critic networks.

Figure 2 presents the basic structure of the implementation of the reinforcement learning agent. The process in the agent can be classified into three major simultaneous processes. In rule one, the actor-network $\mu(s|\theta^{\mu})$ in the RL agent receives a state s_t from the environment, predicts an action a_t based on this state and using the policy learned, and passes it

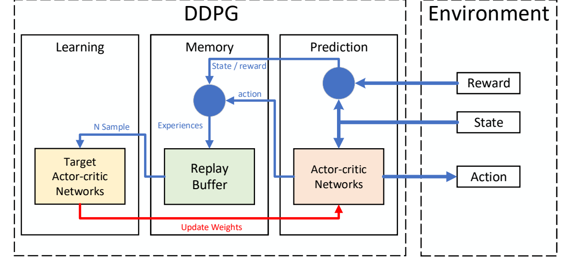


Figure 2: DDPG agent Setup Structure.

(with additional noise $\mathcal{N}$) to the environment for execution, as shown in Equation 1.

$$a_t = \mu(s|\theta^{\mu}) + \mathcal{N}_t \quad (1)$$

Consequently, both the state received from the environment and the predicted action are fed to the critic-network $Q(s,a|\theta^{Q})$ to estimate an action-state value function $Q(s,a)$. Once the agent ends up in a new state s_{t+1}, an experience tuple consisting of state, action, reward, and follower state (s_t, a_t, r_t, s_{t+1}) is collected in a finite-sized cache known as replay buffer. When the replay buffer is full, it discards the oldest samples and stores new ones. In rule three, a minibatch sample of size N from the collected experience in the replay buffer is delivered to a copy of the actor-critic networks. The copy architecture is known as target-actor $\mu'(s|\theta^{\mu})$ and target-critic $Q'(s,a|\theta^{Q})$ networks. Those networks are used to predict for each sample $i \in N$ of the minibatch an output y_i based on the Equation 2 (Lillicrap et al., 2016):

$$y_i = r_i + \gamma Q'(s_{i+1}, \underbrace{\mu'(s_{i+1}|\theta^{\mu'})}_{a_{i+1}}|\theta^{Q'}) \quad (2)$$

Then, the predicted values are used to calculate the loss function in Equation 3 (Lillicrap et al., 2016) and update the critic network weights by minimizing the loss. While the actor policy is updated using the sampled policy gradient shown in Equation 4:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2 \quad (3)$$

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \nabla_a Q(s,a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i} \quad (4)$$

A feedforward, fully-connected Multi-Layer-Perceptron ANN structure with two hidden layers is used for both actor and critic networks. Rectified Linear Unit (ReLU) activation function is applied to all neurons in the hidden layers. The output layer is then either the action the agent can perform or the state-action value function. Tangent-hyperbolic (tanh) activation function is employed to neurons in the output layers of both structures.

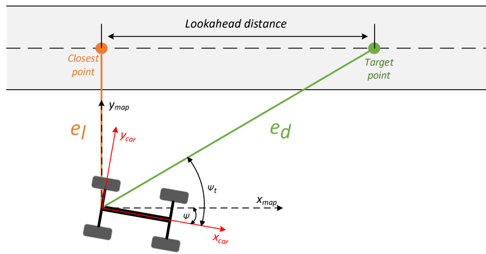


Figure 3: Path following task setup; the agent is trained to follow a given path by controlling its steering. Besides the distance to the target e_d, the cross-track-error e_l is measured and minimized.

The actor-network is used to learn the deterministic policy of the DDPG agent, where an explicit action a is determined directly based on the car observation data and the target point. The critic network learns state-action value function Q based on the current state and the action selected in it. The action was not included until the second hidden layer of the critic network. Both the number of neurons in the hidden layers of the ANN, besides the learning rate, are arbitrarily selectable hyperparameters. The Ornstein-Uhlenbeck (OU) noise process, added to the action to guarantee the agent's exploration, is defined by the parameters $\theta = 0.15$, $\mu = 0$, and $\sigma = 0.2$.

3.3 Path Following Setup

Besides the distance to the target, the agent learns to minimize the cross-track error to achieve the best match between the driving path and the desired one. Figure 3 simulates the agent in one of the possible states. The agent is supposed to drive at a constant speed and interact with an anonymous environment to explore it, attempting to reach a target point. While the agent is moving, the desired target will be updated continuously with all other parameters, and based on the reward function, it receives either a reward or a penalty. In addition, the agent is modifying its orientation by controlling its steering based on the reward it receives.

Figure 4a shows the middle points of the path that will be used to train the agent. Using several points to train the agent will enhance its performance when tested on other trajectories. The agent will learn how to approach points by steering left and points on his longitudinal axis. Still, due to the limitation of the map, the agent might fail on the tracks that have the right turns. The desired speed is estimated based on the orientation error between the agent and the target point. Figure 4b shows the relationship between the target speed and the orientation error.

Table 1: Observation data provided through the environment.

Symbol	Description	Min	Max
x [m]	Car position on x axis	-0.1	6.30
y [m]	Car position on y axis	-0.1	4.50
ψ [rad]	Car orientation yaw	$-\pi$	$+\pi$
v_x [m/s]	Car speed on x axis	-0.0	+0.8
v_y [m/s]	Car speed on y axis	-0.0	+0.1
x_t [m]	Target position on x axis	-0.1	6.30
y_t [m]	Target position on y axis	0.1	4.50
ψ_t [rad]	Target orientation yaw	$-\pi$	$+\pi$
v_t [m/s]	Target linear speed	0.0	+0.8

Table 2: List of error measured by the environment step function.

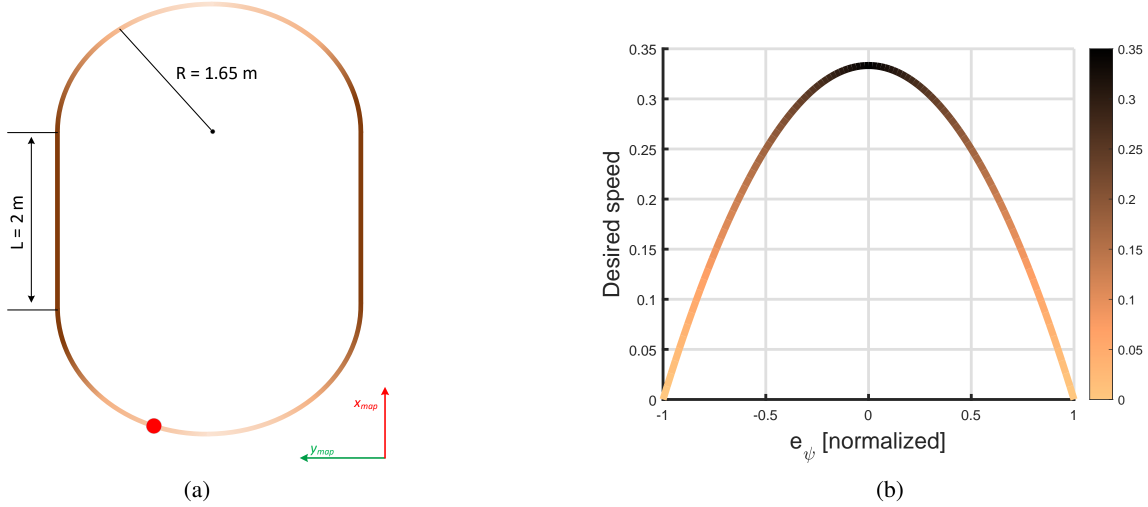
Symbol	Description
e_d	Distance to target
e_l	Cross track error
e_ψ	Orientation error

The RL agent demands to get all associated information about the current state of the environment to calculate the control components and accomplish the mission. The environment provides various continuous vehicle state variables and sensors data. However, the main goal is to train the agent with as few as possible low-dimensional data. The symbols of state variables distinguished as necessary, including a short description and the value range used for data preprocessing, are shown in Table 1. (x, y) describes the current position of the vehicle at the midpoint of the rear axle, where (v_x, v_y) are the linear vehicle speed vector components. ψ_a defines the rotational movement of the vehicle around the vertical axis of the vehicle coordinate system. (x_t, y_t, ψ_t) are the target point and orientation respectively. v_T is the target speed.

Furthermore, various error measures are available, which will be used to design the RL agents' reward function. Table 2 shows a list of error quantities calculated in the state. e_d is longitude to the Target position. In contrast, the cross-track error e_l describes the deviation from the nominal to the actual path, and the yaw angle error e_ψ describes the deviation in the alignment of the vehicle. All observation quantities and errors are normalized.

One of the key advantages of using DDPG is its ability to operate over continuous action spaces. In the studied challenge, the agent learns a continuous steering angle, which can assume any value between $[-0.78, +0.95]$ rad. Table 3 shows the value range for action space

Rewards are calculated each time step t after taking action a. It can be viewed as a feedback flag that evaluates how good taking action a in state s is. In this



(a) (b)

Figure 4: (a) The path used to train the RL-agents, (b) beside the function used to predict the desired speed, while the actual speed value before normalization can be found in Table 1.

Table 3: RL-agents continuous action space.

Symbol	Description	Min	Max
δ [rad]	Steering Command	-0,78	+0,95
v [m/s]	Speed Command	-0.0	+0,8

task, two main errors should be minimized in each step: the vehicle's orientation error and the deviation between the target and actual path of the vehicle, since the vehicle should deviate as little as possible from the planned line of the path planner. Therefore, the reward function should evaluate both e_ψ and e_l.

$$r(t) = r(e_\psi) + r(e_l) \tag{5}$$

However, the cross-track error needs to be limited to end the training episode and reset the agent once exceeding it.

$$r = \begin{cases} -10 & \text{if } e_d > D \\ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{((e_\psi + e_l) - \mu)^2}{2\sigma^2}\right) & else \end{cases} \tag{6}$$

A Gaussian distribution is used to determine the rewards component, shifted by -1 with mean value $\mu = 0$ and standard deviation $\sigma = 0.2$.

The network structure mentioned in 3.3, and Table 4 presents the number of neurons used in the hidden layers for both networks. The learning process applied is Adam with a learning rate of 10^{-4} and 10^{-3} for the actor and critic networks, respectively. A discount factor of $\gamma = 0.99$, and for soft target updates $\tau = 10^{-3}$ were used. The random noise process was added to the action output value to guarantee the agent's exploration during the whole training procedure. It is defined by the parameters $\theta = 0.15$, $\mu = 0$, and $\sigma = 0.2$.

Table 4: Number of neurons used in hidden layers for the actor and critic networks.

	h_{1a}	h_{2a}	h_{1c}	h_{2c}
Architecture 1	400	300	400	300

4 EVALUATION

The training is set to 120 000 steps and elaborated as a continuous task. i.e., no maximum number of steps per episode is defined. The episode ends only when the cross-track-error is higher than a specific value (D = 20 cm). When the agent exceeds the defined cross-track error, the learning process pauses, and the agent position is set back close to the track again. Only then does the training resume. The maximum allowed deviation of the target path is chosen based on the map size and the function's capabilities that reset the agent position.

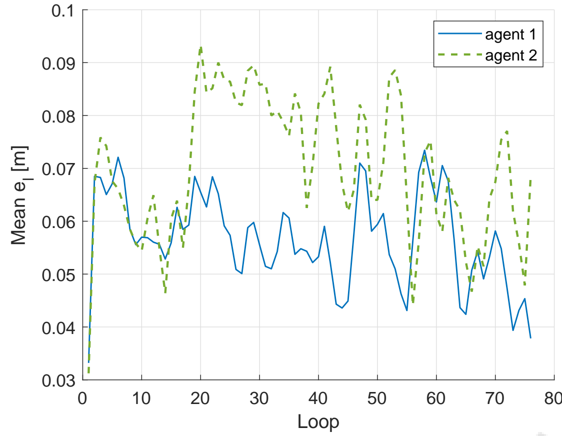
4.1 Reward Function Evaluation

Considering Figure 3, the setup includes three main errors. The euclidean distance to the target point e_d, the cross-track error e_l, and the orientation error e_ψ. In this experiment, we will study the effect of combining e_l with e_ψ in the reward function $R(e_\psi, e_l)$, and compare the result with using reward function $R(e_\psi)$ that only consider the e_ψ.

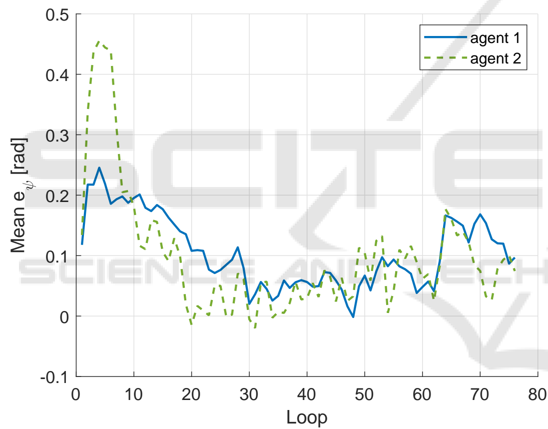
Table 5 presents a quick review of the setup used to train the agents. For both agents, a fixed lookahead offset of 60 cm is defined, the same network structure, shown in Table 4 is used. Furthermore, both used the observation space shown in Table 1. The agents were tasked to obtain proper steering policies.

Table 5: Comparison, of setup used to train RL-agents to verify the reward function.

	Observation	Action	Reward	Lookahead	Network
Agent 1	Table 1	δ	$R(e_\psi, e_l)$	60 [cm]	Table 4
Agent 2	Table 1	δ	$R(e_\psi)$	60 [cm]	Table 4



(a)



(b)

Figure 5: Comparison of the two RL-agents performance during training for 76 loops with lookahead offset of 60 cm; (a) the mean cross-track error over each loop of the training. (b) the mean orientation error vs. training loops.

Both agents were trained for 76 loops around the path. In Figure 5, the mean orientation error e_ψ, and mean cross-track error e_l are plotted vs. loops over the whole training for both agents. As we can spot out from the figure, integrating the cross-track error in the reward function enhanced the agent's ability to drive closer to the target path without affecting the orientation error. Nevertheless, both agents could drive around the path for a full loop by the end of the training.

In order to evaluate the performance of both agents, they were both tested after the end of the training. The idea was to let them drive around the path for several loops and measure the Root-Mean-Square (RMS) for the orientation error and the cross-track error and compare the results of a quantitative perspective. Then, in the main time, observe the changes of both errors during one loop drive.

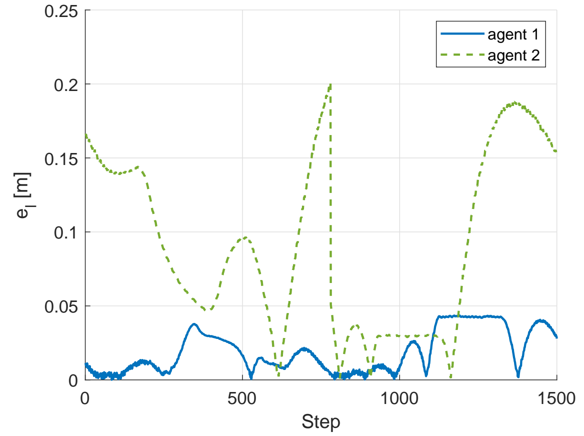


Figure 6: Comparison of the two RL-agents performance during testing for one loop with lookahead offset of 60 cm; the cross-track error over one loop of the testing.

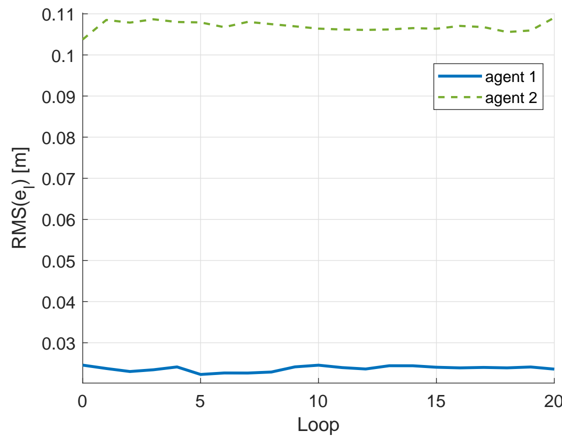
Table 6: Number of hidden layers neurons in each RL-agent.

	h_{1a}	h_{2a}	h_{1c}	h_{2c}
Architecture 1	400	300	400	300
Architecture 2	600	450	600	450

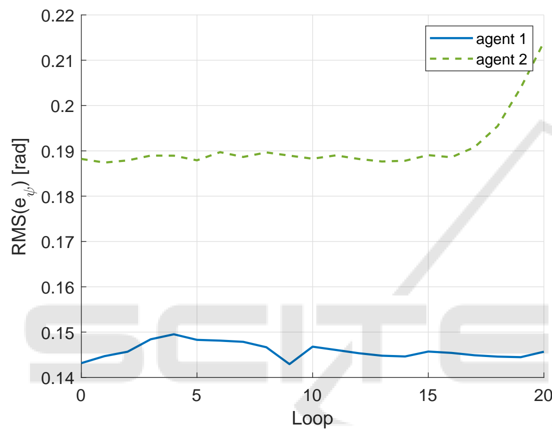
Figure 6 plots the variations of the cross-track error during the second loop of the test. In the figure, it is shown that agent 1 showed more steady performance in keeping the car close to the path than agent 2. Moreover, agent 2 failed to finish one loop around the track, and a reset position was needed to put it back close to the track. Figure 7 shows the RMS error over the 20 loops of the testing for both agents. As a result, we can assess the performance of both agents over the whole testing period.

4.2 Neural Network Evaluation

In this experiments, different agents with various neural network architectures were trained to learn the steering policy. Two of them outperformed the other ones. In the following, the victorious agent's results will be compared. Table 6 shows the number of hidden layers neurons in each RL-agent.



(a)



(b)

Figure 7: Comparison of the two RL-agents performance during testing for 20 loops with lookahead offset of 60 cm; (a) the RMS cross-track error over 20 loops during testing. (b) the RMS orientation error vs. 20 loops.

Both agents learned on the same reward function $R(e_\psi, e_l)$, and were trained for 76 loops with a fixed lookahead offset of 60 cm. Furthermore, both used the observation space shown in Table 1. Finally, the agents were tasked to obtain proper steering policies. In Figure 8, the mean cross-track error e_l is plotted vs. loops over the whole training for both agents. As we can spot out from the figure, the agent used architecture 2 could remain a small cross-track error by the end of the training. Nevertheless, both agents could drive around the path for a full loop by the end of the training.

In order to evaluate the performance of both agents, they were both tested after the end of the training. The idea was to let them drive around the path for several loops and measure, at each loop, the RMS for the orientation error, as well as the cross-track error and compare the results of a quantitative perspective.

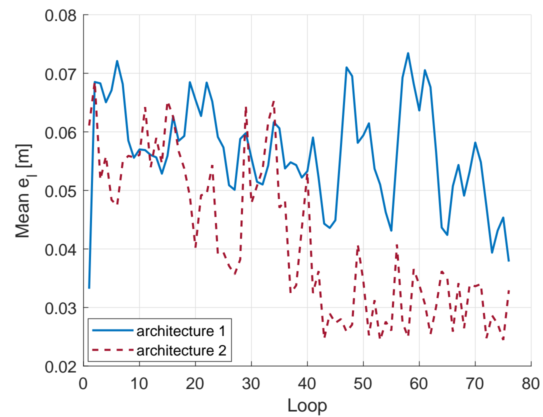


Figure 8: Comparison of the two RL-agents performance during training for 76 loops with lookahead offset of 60 cm; the mean cross-track error over each loop of the training.

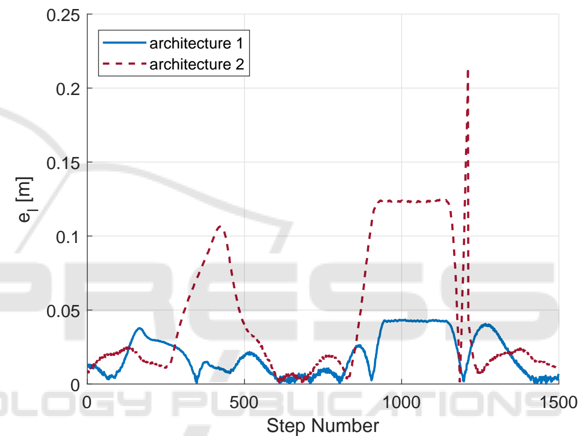


Figure 9: Comparison of the two RL-agents performance during testing for one loop with lookahead offset of 60 cm; the cross-track error over one loop of the testing.

In the main time, observe the changes of both errors during one loop drive.

Figure 9 plot the variations of the cross-track error during the second loop of the test. As can be noticed from the Figure, the agent with architecture 1 showed more steady performance in keeping the car close to the path than agent 2. Moreover, the agent with architecture 2 failed to finish one loop around the track. A reset position was needed to put it back close to the track at the final curvature of the path. Figure 10 shows the RMS(e_l) over the 6 loops of the testing for both agents. As a result, we can assess the performance of both agents over the whole testing period.

The training of the agent with architecture 2 was repeated with a double number of training steps. Still, it was not able to show a better performance than the presented one. For this reason, we will not consider it in any further evaluation.

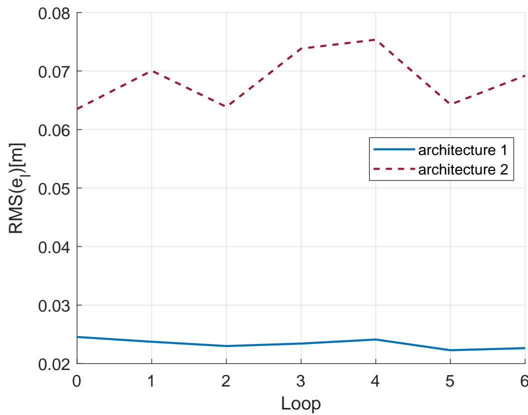


Figure 10: Comparison of the two RL-agents performance during testing for 6 loops with lookahead offset of 60 cm; the RMS cross-track error over 6 loops during testing.

4.3 Further Evaluation

The presented experiments confirmed the importance of rewarding the cross-track error besides the orientation error in the path tracking tasks. Agent 1 showed an exceptional performance during testing the agent. However, we still need to prove how dynamic the trained agent is for changes, e.g., the lookahead distance. For this reason, the agent will be tested for different lookahead distances (20, 40, 80, 100) cm. By modifying the lookahead distance, we can analyze the agent's steering policy strength to respond for more extensive and smaller steering values with keeping the cross-track error as small as possible.

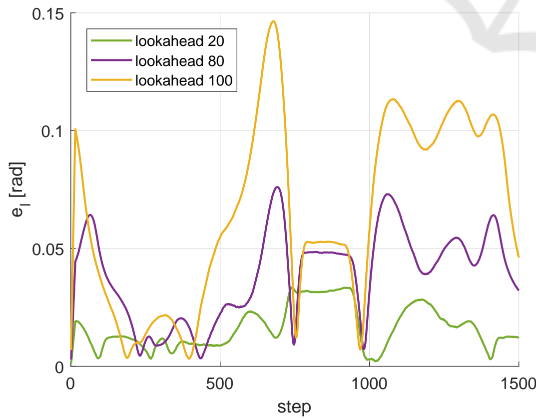


Figure 11: Testing RL-agents performance For different lookahead offset values;the cross-track error over one loop of the testing.

Figure 11 shows the measured errors over one loop during the proposed test, for some elected lookahead offset of (20, 80, 100) cm, quantitatively. Thus, the agent was able to drive around the track without exceeding the maximum tolerated deviation ($D = 20cm$). Moreover, the orientation error variation is also in an acceptable range. However, to evaluate the test more reasonably, the test was held for several loops, and the mean error (e_ψ and e_d) over each loop, for the whole testing period, is measured and plotted in Figure 12.

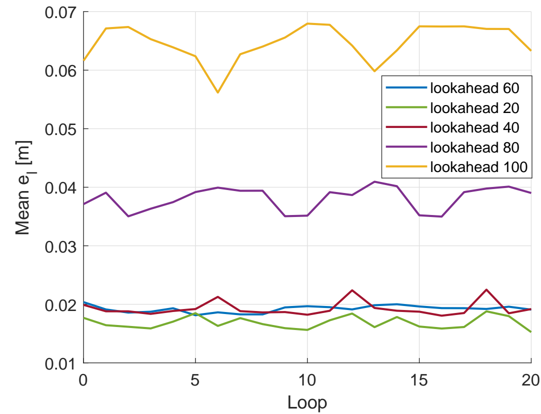


Figure 12: Comparison of the RL-agents performance during testing for 20 loops with different lookahead offset; (a) the RMS cross-track error over 20 loops during testing.

After quantitatively evaluating the agent, it is time to evaluate it qualitatively. In Figure 13, The routes the agent drove during testing for one loop with different lookahead offset are plotted as well as the optimal route. As we can see, the routes are very close to each other. In order to summarize this plot information, the mean and the Standard Deviation (SD) for each route are shown in Table 7.

5 CONCLUSIONS

In the context of this paper, the state-of-the-art deep reinforcement learning algorithm DDPG was selected

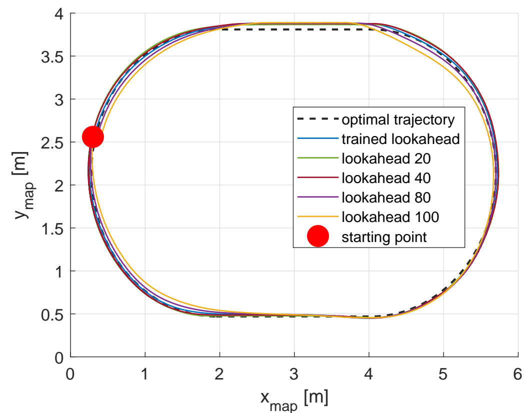


Figure 13: Agent's driven route during testing, with different lookahead testing, together with the optimal path.

Table 7: Mean and Standard Deviation values for testing the RL-agent with different lookahead offset.

Lookahead	**60**	**20**	**40**	**80**	**100**
Mean	0.0195	0.0164	0.0188	0.0391	0.0671
SD	0.0141	0.0090	0.0110	0.0190	0.0398

as an end-to-end learning approach to take over the comprehensive steering control for an autonomous vehicle. The proposed agent has been trained and tested in the AutoMiny-Gazebo environment, which implements a realistic model of the AutoMiny model car. The target point was updating continuously in each iteration during training. The aim was to encourage the agent to follow a pre-defined path with a minimum cross-track error. A continuous state space includes the agent position, orientation, speed, and the target point coordinates, and the desired orientation was employed. Both cross-track error and orientation error were combined in the reward function. Updating the target points during training made the agent gain more experience and drove a complete loop around the track after 80 training loops. Although the achieved results seem very encouraging, more testing is necessary to validate the agent's ability to follow different paths with more diverse route profiles. Moreover, enhance the approach to include computing the optimal velocity of the vehicle depending on the path and deploying the agent in a real platform.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Bundesministerium fur Verkehr und digitale Infrastruktur (BMVI) in Germany as part of the Shuttles&Co project, within the Automatisiertes, Vernetztes Fahren (AVF) program.

REFERENCES

Alomari, K., Carrillo Mendoza, R., Sundermann, S., Göhring, D., and Rojas, R. (2020). Fuzzy logic-based adaptive cruise control for autonomous model car. In *ROBOVIS*.

Alomari, K., Sundermann, S., Göhring, D., and Rojas, R. (in press). Design and experimental analysis of an adaptive cruise control. *Springer CCIS book series*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Calzolari, D., Schürmann, B., and Althoff, M. (2017). Comparison of trajectory tracking controllers for autonomous vehicles. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8.

Chan, C.-Y. (2017). Advancements, prospects, and impacts of automated driving systems. *International Journal of Transportation Science and Technology*, 6(3):208 – 216. Safer Road Infrastructure and Operation Management.

Hall, J., Rasmussen, C. E., and Maciejowski, J. (2011). Reinforcement learning with reference tracking control in continuous state spaces. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 6019–6024.

Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., and Nashashibi, F. (2018). End-to-end race driving with deep reinforcement learning. *CoRR*, abs/1807.02371.

Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., and Shah, A. (2018). Learning to drive in a day. *CoRR*, abs/1807.00412.

Li, D., Zhao, D., Zhang, Q., and Chen, Y. (2019). Reinforcement learning and deep learning based lateral control for autonomous driving [application notes]. *IEEE Computational Intelligence Magazine*, 14(2):83–98.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Martinsen, A. B. and Lekkas, A. M. (2018). Curved path following with deep reinforcement learning: Results from three vessel models. In *OCEANS 2018 MTS/IEEE Charleston*, pages 1–8.

Schmidt, M., Bünger, S., and Chen, Y. (2019). Autominy-simulator mit gazebo.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Wang, S., Jia, D., and Weng, X. (2018). Deep reinforcement learning for autonomous driving. *CoRR*, abs/1811.11329.