ACTA IMEKO
ISSN: 2221-870X
September 2021, Volume 10, Number 3, 7 - 14

Vision-based reinforcement learning for lane-tracking control

András Kalapos[1], Csaba Gór[2], Róbert Moni[3], István Harmati[1]

[1] *BME, Dept. of Control Engineering and Information Technology, Budapest, Hungary*
[2] *Continental ADAS AI, Budapest, Hungary*
[3] *BME, Dept. of Telecommunications and Media Informatics, Budapest, Hungary*

ABSTRACT
The present study focused on vision-based end-to-end reinforcement learning in relation to vehicle control problems such as lane following and collision avoidance. The controller policy presented in this paper is able to control a small-scale robot to follow the right-hand lane of a real two-lane road, although its training has only been carried out in a simulation. This model, realised by a simple, convolutional network, relies on images of a forward-facing monocular camera and generates continuous actions that directly control the vehicle. To train this policy, proximal policy optimization was used, and to achieve the generalisation capability required for real performance, domain randomisation was used. A thorough analysis of the trained policy was conducted by measuring multiple performance metrics and comparing these to baselines that rely on other methods. To assess the quality of the simulation-to-reality transfer learning process and the performance of the controller in the real world, simple metrics were measured on a real track and compared with results from a matching simulation. Further analysis was carried out by visualising salient object maps.

Section: RESEARCH PAPER

Keywords: Artificial intelligence, machine learning, mobile robot, reinforcement learning, simulation-to-reality, transfer learning

Citation: András Kalapos, Csaba Gór, Róbert Moni, István Harmati, Vision-based reinforcement learning for lane-tracking control, Acta IMEKO, vol. 10, no. 3, article 4, September 2021, identifier: IMEKO-ACTA-10 (2021)-03-04

Section Editor: Bálint Kiss, Budapest University of Technology and Economics, Hungary

Received January 17, 2021; **In final form** September 22, 2021; **Published** September 2021

Corresponding author: András Kalapos, e-mail: andras.kalapos.research@gmail.com

1. INTRODUCTION

Reinforcement learning has been used to solve many control and robotics tasks. However, only a handful of papers have been published that apply this technique to end-to-end driving [1]-[7], and even fewer studies have focused on reinforcement learning-based driving, trained only in simulations and then applied to real-world problems. Generally, bridging the gap between simulation and the real world is an important transfer-learning problem related to reinforcement learning, and it is an unresolved task for researchers.

Mnih et al. [1] proposed a method to train vehicle controller policies that predict discrete control actions based on a single image of a forward-facing camera. Jaritz et al. [2] used WRC6, a realistic racing simulator, to train a vision-based road-following policy. They assessed the policy's generalisation capability by testing it on previously unseen tracks and on real driving videos in an open-loop configuration; but their work did not extend to an evaluation of real vehicles in closed-loop control. Kendall et al. [3] demonstrated real-world driving by training a lane-following policy exclusively on a real vehicle under the supervision of a safety driver. Shi et al. [4] presented research that involved training reinforcement learning agents in Duckietown, in a similar way to that presented here; however, the focus was mainly on presenting a method that explained the reasoning behind the trained agents rather than the training methods. Also similar to the present study, Balaji et al. [5] presented a method for training a road-following policy in a simulator using reinforcement learning and tested the trained agent in the real world, yet their primary contribution is the DeepRacer platform rather than an in-depth analysis of the road-following policy. Almási et al. [7] also used reinforcement learning to solve lane following in the Duckietown environment, but their work differs from the present study in the use of an off-policy reinforcement learning algorithm (deep Q-networks (DQNs) [8]); in this study an on-policy algorithm (proximal policy optimization [9]) is used, which achieves significantly better sample efficiency and shorter training times. Another important difference is that Almási et al. applied hand-crafted colour threshold-based segmentation to the input images, whereas the method presented here takes the 'raw' images as inputs, which allows for a more robust real performance.

This paper is an extended version of the authors' original contribution [10]. It includes the results of the 5th AI Driving Olympics [11] and aims to improve the description of the methods. In both works, vision-based end-to-end reinforcement learning relating to vehicle control problems is studied and a solution is proposed that performs lane following in the real world, using continuous actions, without any real data provided by an expert (as in [3]). Also, validation of the trained policies in both the real and simulated domains is conducted.

The training and evaluation code for this paper is available on GitHub[1].

2. METHODS

In this study, a neural-network-based controller was trained that takes images from a forward-looking monocular camera and produces control signals to drive a vehicle in the right-hand lane of a two-way road. The vehicle to be controlled was a small differential-wheeled mobile robot, a Duckiebot, which is part of the Duckietown ecosystem [11], a simple and accessible platform for research and education on mobile robotics and autonomous vehicles. The primary objective was to travel as far as possible within a given time without leaving the road. Lane departure was allowed but not preferred. Although the latest version of the Duckiebot is equipped with wheel encoders, for this method, the vehicle was solely reliant on data from the robot's forward-facing monocular camera.

2.1. Reinforcement learning algorithm

In reinforcement learning, an agent interacts with the environment by taking a_t action, then the environment returns s_{t+1} observation and r_{t+1} reward. The agent computes the next a_{t+1} action based on s_{t+1} and so on. The policy is the parametric controller of the agent, and it is tuned during the reinforcement learning training. Sequences of actions, observations and rewards (τ trajectories) are used to train the parameters of the policy to maximise the expected reward over a finite number of steps (agent–environment interactions). For vehicle control problems, the actions are the signals that control the vehicle, such as the steering and throttle, and the observations are the sensor data relating to the environment of the vehicle, such as the camera, lidar data or higher-level environment models. In this research, the observations were images from the robot's forward-facing camera, and the actions were the velocity signals for the two wheels of the robot.

Policy optimisation algorithms are on-policy reinforcement learning methods that optimise the parameters of the $\pi_\theta(a_t|s_t)$ policy based on the a_t actions and the r_t reward received for them; θ denotes the trainable parameters of the policy. On-policy reinforcement learning algorithms optimise the $\pi_\theta(a_t|s_t)$ policy based on trajectories in which the actions have been computed by $\pi_\theta(a_t|s_t)$. In contrast, off-policy algorithms (such as DQNs [8]) compute actions based on the estimate of the action-value function of the environment, which they learn using data from a large number of (earlier) trajectories, making these algorithms less stable in some environments. In policy optimisation algorithms, the $\pi_\theta(a_t|s_t)$ policy is stochastic, and in the case of deep reinforcement learning, it is implemented by a neural network, which is updated using a gradient method. The policy is stochastic because, instead of computing the actions directly,

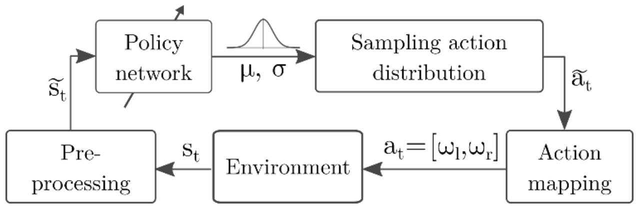


Figure 1. Illustration of the policy architecture with the notations used. The agent is represented jointly by the 'Policy network' and 'Sampling action distribution' blocks; s_t: 'raw' observation, $\tilde{s}_t$: pre-processed observation, $\tilde{a}_t$: predicted action, a_t: post-processed action.

the policy network predicts the parameters of a probability distribution (see μ and σ in Figure 1) that is sampled to acquire the $\tilde{a}_t$ predicted actions (here, predicted refers to this action being predicted by the policy).

In the present study, to train the policy, the proximal policy optimization algorithm [9] was used because of its stability, sample-complexity and ability to take advantage of multiple parallel workers.

Proximal policy optimization performs the weight updates using a special loss function to keep the new policy close to the old, thereby improving the stability of the training. Two loss functions were proposed by Schulman et al. [9]:

$$\mathfrak{L}_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}\left[\min\left(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \quad (1)$$

$$\mathfrak{L}_{\text{KLPEN}}(\theta) = \hat{\mathbb{E}}\left[\rho_t(\theta)\hat{A} - \beta\,\text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\right]\right], \quad (2)$$

where $\text{clip}(\cdot)$ and $\text{KL}[\cdot]$ refer to the clipping function and the Kullback–Leibler (KL) divergence, respectively, while $\hat{A}$ is calculated as the generalised advantage estimate [12]. In these loss functions, ϵ is usually a constant in the $[0.1, 0.3]$ range, while β is an adaptive parameter, and

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \quad (3)$$

An open-source implementation of proximal policy optimization from RLlib [13] was used, which performs the gradient updates based on the weighted sum of these loss functions. The pseudo code and additional details for the algorithm are provided in the Appendix.

2.2. Policy architecture

The controller policy was realised by a shallow (4-layer) convolutional neural network. Both the policy and the value network used the architecture presented by Mnih et al. [1], with the only difference being the use of linear activation in the output of the policy network. No weights were shared between the policy and the value network. This policy is considered to be end-to-end because the only learning component is the neural network, which directly computes actions based on observations from the environment.

Some pre- and post-processing was applied to the observations and actions, but these only performed very simple transformations (explained in the next paragraph and Section 2.3). The aim of these pre- and post-processing steps was to transform the s_t observations and a_t actions into representations that enabled faster convergence without losing

[1] https://github.com/kaland313/Duckietown-RL (Accessed 23 September 2021)

any important features in the observations or restricting necessary actions.

The input of the policy network consisted of the last three observations (images) scaled, cropped and stacked (along the depth axis). The observations returned by the environment (s_t on Figure 1) were 640 × 480 (width, height) RGB images, the top third of which mainly showed the sky, which was therefore cropped. The cropped images were then scaled down to 84 × 84 resolution (note the uneven scaling), which were then stacked along the depth axis, resulting in 84 × 84 × 9 input tensors ($\widetilde{s}_t$ in Figure 1). The last three images were stacked to provide the policy with information about the robot's speed and acceleration.

Multiple action representations were experimented with (see Section 2.3). Based on these representations, the policy outputs $\widetilde{a}_t$ predicted an action vector of two or a scalar value that controlled the vehicle. The policy was stochastic, and the output of the neural network therefore produced the μ and $\log\sigma$ parameters of a multivariate diagonal normal distribution. During training, this distribution was sampled to acquire the $\widetilde{a}_t$ actions, which improved the exploration of the action space. During these evaluations, the sampling step was skipped by using the predicted μ mean value as the $\widetilde{a}_t$ policy output.

2.3. Action representations

The action mapping step transformed the $\widetilde{a}_t$ predicted actions, which could be implemented using many representations, to $a_t = [\omega_l, \omega_r]$ wheel velocities (see Figure 1). The vehicle to be controlled was a differential-wheeled robot; the most basic action representation was therefore to directly compute the angular velocities of the two wheels as continuous values in the $\omega_{l,r} \in [-1; 1]$ range (where 1 and -1 corresponded to forward and backward rotation at full speed). However, this action space allowed for actions that were not necessary for the manoeuvres examined in this paper. Moreover, as the reinforcement learning algorithm ruled out unnecessary actions, exploration of the action space was potentially made more difficult, and the number of steps required to train an agent was therefore increased.

Several methods can be used to constrain and simplify the action space, such as discretisation, clipping some actions or mapping to a lower-dimensional space. Most previous studies [1],[2],[5],[7] have used discrete action spaces, thus the neural network in these policies selected one from a set of hand-crafted actions (steering, throttle combinations), while Kendall et al. [3] utilised continuous actions, as has been used in this study.

In order to test the reinforcement learning algorithm's ability to address general tasks, multiple action mappings and simplifications of the action space were experimented with. These are described in the following paragraphs.

Wheel velocity: Wheel velocities were a direct output of the policy; $a_t = [\omega_l, \omega_r] = \widetilde{a}_t$, therefore $\omega_{l,r} \in [-1; 1]$.

Wheel velocity - positive only: Only positive wheel velocities were allowed because only these were required to move forward. Values predicted outside the $\omega_{l,r} \in [0; 1]$ interval were clipped: $a_t = [\omega_l, \omega_r] = \mathrm{clip}(\widetilde{a}_t, 0{,}1)$.

Wheel velocity - braking: Wheel velocities were still only able to fall within the $\omega_{l,r} \in [0; 1]$ interval, but the predicted values were interpreted as the amount of braking from the maximum speed. The main differentiating factor from the 'positive only' option was the bias towards moving forward at full speed:
$a_t = [\omega_l, \omega_r] = \mathrm{clip}(1 - \widetilde{a}_t, 0{,}1)$.

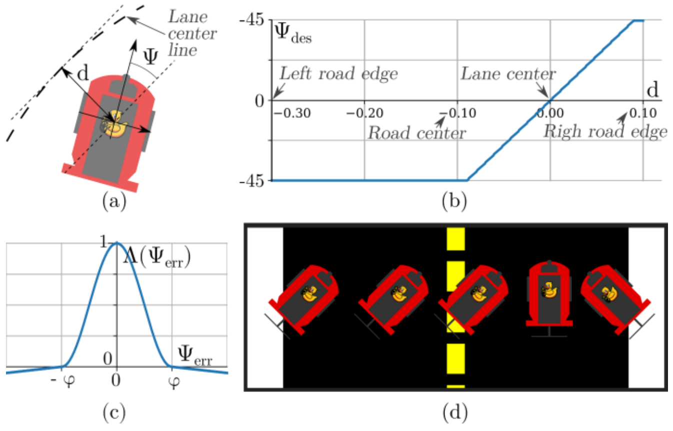


Figure 2. Explanation of the proposed orientation reward: (a) explains Ψ, d, (b) shows how the desired orientation depends on the lateral error, (c) shows the $\Lambda(x)$ function and (d) provides some examples of desired configurations.

Steering: Predicting a scalar value that was continuously mapped to combinations of wheel velocities. The 0.0 scalar value corresponds to moving straight (at full speed), while −1.0 and 1.0 refer to turning left or right with one wheel completely stopped and the other going at full speed. Intermediate values are computed using linear interpolation between these values. The speed of the robot is always maximal for a particular steering value. Below is the formula that implements this action mapping: $a_t = [\omega_l, \omega_r] = \mathrm{clip}([1 + \widetilde{a}_t, 1 - \widetilde{a}_t], 0{,}1)$.

2.4. Reward shaping

The reward function is a fundamental element of every reinforcement learning problem, as it serves the important role of converting a task from a textual description to a mathematical optimisation problem. The primary objective for the agent is to travel as far as possible within a given time in the right-hand lane; therefore, two rewards that promote this behaviour were proposed.

Distance travelled: The agent's reward was directly proportional to the distance it moved along the right-hand lane at each step. Only longitudinal motion was counted and only if the robot stayed in the right-hand lane.

Orientation: The agent was rewarded if it was facing and moving in the desired orientation, which was determined based on its lateral position. In simple terms, it received the largest reward if it faced towards the centre of the right-hand lane (some example configurations are shown in Figure 2 d). A term proportional to the angular velocity of the faster moving wheel was also added to encourage fast motion.

This reward was calculated as $r = \lambda_\Psi\, r_\Psi(\Psi, d) + \lambda_v\, r_v(\omega_l, \omega_r)$, where $r_\Psi(\cdot), r_v(\cdot)$ are the orientation and velocity-based components (explained below), while the λ_Ψ, λ_v constants scale these to [-1,1]. Ψ, d are the orientation and lateral error from the desired trajectory, which is the centreline of the right-hand lane (see Figure 2 a).

The orientation-based term was calculated as $r_\Psi(\Psi, d) = \Lambda(\Psi_{err}) = \Lambda(\Psi - \Psi_{\mathrm{des}}(d))$, where $\Psi_{\mathrm{des}}(d)$ is the desired orientation calculated using the lateral distance from the desired trajectory (see Figure 2 b for the illustration of $\Psi_{\mathrm{des}}(d)$). The Λ function achieves the promotion of the $|\Psi_{err}| < \varphi$ error, while an error larger than φ leads to a small negative reward (a plot of $\Lambda(x)$ is shown in Figure 2 c):

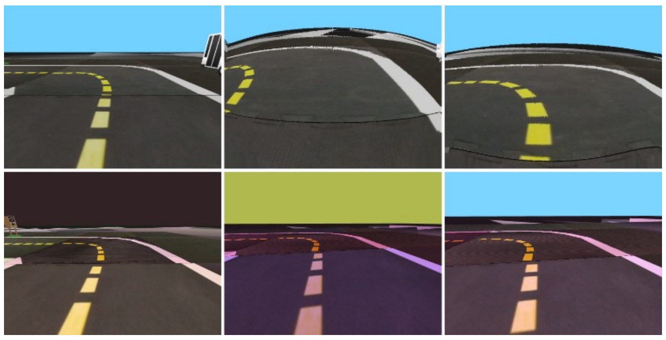


Figure 3. Examples of domain randomised observations.

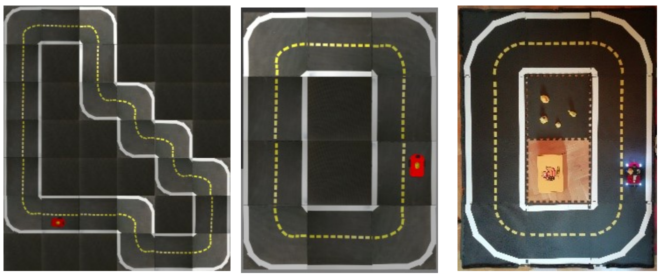


Figure 4. a) Test track used for simulated reinforcement learning and baseline evaluations; b) and c) real and simulated test track used for the evaluation of the simulation-to-reality transfer.

$$\Lambda(x) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\left(\pi\frac{x}{\varphi}\right) & \text{if } -1 \leq x \leq 1 \\ \varepsilon\left(1 - \left|\frac{x}{\varphi}\right|\right) & \text{otherwise} \end{cases}, \quad (4)$$

where the $\varepsilon \in [10^{-1}, 10^{-2}]$ and $\varphi = 50°$ hyperparameters are selected arbitrarily.

The velocity-based component was calculated as $r_v(\omega_l, \omega_r) = \max(\omega_l, \omega_r)$ to reward an equally high-speed motion in both straight and curved sections. In the curved sections, only the outer wheel was able to rotate at maximal speed, while on a straight road, both wheels were able to do so.

2.5. Simulation-to-reality transfer

To train the agents, an open-source simulation of the Duckietown environment was used [14]. This simulation models certain physical properties of the real environment accurately (dimensions of the robot, camera parameters, dynamic properties, etc.), but several other effects (textures, objects at the side of the road) and light simulation are less realistic (e.g. compared to modern computer games). These inaccuracies create a gap between simulation and reality that makes it challenging for any reinforcement learning agent to be trained only in simulation but operate in reality.

To bridge the simulation-to-reality gap and to achieve the generalisation capability required for real performance, domain randomisation was used. This involves training the policy in many different variants of a simulated environment by varying lighting conditions, object textures, the camera, vehicle dynamics parameters and road structures (see Figure 3 for examples of domain randomised observations). In addition to the 'built-in' randomisation options of Gym-Duckietown, this study used a diverse set of maps to train on in order to further improve the agent's generalisation capability.

2.6. Collision avoidance

Collision avoidance with other vehicles greatly increases the complexity of the lane-following task. These problems can be solved in different ways, for example, by overtaking or following at a safe distance. However, the sensing capability of the vehicle and the complexity of the policy determine the solution it can learn. Images from the forward-facing camera of a Duckiebot only have a 160 ° horizontal field of view; therefore, the policy controlling the vehicle has no information about objects moving next to or behind the robot. For simplicity, in this study, the same convolutional network for collision avoidance as for lane following was used, which does not feature a long short-term memory cell or any other sequence modelling component (in contrast to [2]). For these reasons, it is unable to plan long manoeuvres, such as overtaking, which also requires side vision to check when it is safe to return to the right-hand lane. The policy was therefore trained in situations where there was a slow vehicle ahead, and the agent had to learn to perform lane following at full speed until it had caught up with the vehicle in front, at which point it had to reduce its speed and maintain a safe distance to avoid collision.

In these experiments, the *wheel velocity - braking* action representation was used as the policy's output because this allowed the agent to slow down or even stop the vehicle if necessary (unlike the *steering* action). Both the *orientation* and the *distance travelled* reward functions were used to train agents for collision avoidance. The former was supplemented with a term that promoted collision avoidance, while the latter was used unchanged. The simulation used provided a p_{coll} penalty if the safety circles around the two vehicles overlapped. The r_{coll} reward component that promoted collision avoidance was calculated using this penalty. If the penalty decreased because the robot was able to increase its distance from an obstacle, the reward term was proportional to the change in penalty; otherwise, it was 0:

$$r_{\text{coll}} = \begin{cases} -\lambda_{\text{coll}} \cdot \Delta p_{\text{coll}} & \text{if } \Delta p_{\text{coll}} < 0 \\ 0 & \text{otherwise} \end{cases}. \quad (5)$$

This term was added to the *orientation* reward, and it aimed to encourage the policy to increase the distance from the vehicle ahead if it got too close. Collisions were only penalised by terminating the episode without giving any negative rewards.

2.7. Evaluation

To assess the performance of the reinforcement learning-based controller, multiple performance metrics in the simulation were measured and compared against two baselines, one using a classical control theory approach and the other being human driving.

Survival time (t_{survive}) in s: The time until the robot left the road or the duration of an evaluation episode.

Distance travelled in ego-lane (s_{ego}) in m: The distance travelled along the right-hand lane within a fixed time period. Only longitudinal motion was counted; tangential movement therefore counted the most towards this metric.

Distance travelled both lanes (s_{both}) in m: Both the distance travelled along the right-hand-lane within a fixed time period and sections where the agent moved into the oncoming lane counted towards this metric.

Lateral deviation (d_d) in m·s: Lateral deviation from the lane's centreline integrated over the time of an episode.

Orientation deviation (d_Ψ) in rad·s: The robot orientation's deviation from the tangent of the lane centreline integrated over the time of an episode.

Table 1. Comparison of the reinforcement learning agent with two baselines in simulation.

Mean metrics over 5 episodes		RL agent	PD baseline	Human baseline
Survival time in s	↑	15	15	15
Distance travelled both lanes in m	↑	7.1	7.6	7.0
Distance travelled ego-lane in m	↑	7.0	7.6	6.7
Lateral deviation in m ·s	↓	0.5	0.5	0.9
Orientation deviation in rad·s	↓	1.5	1.1	2.8

Time outside ego-lane (t_{out}) in s: Time spent outside the ego-lane.

Even though Duckietown is intended to be a standardised platform, it is still under development, and the official evaluation methods and baselines have not been adopted widely in the research community. The AI Driving Olympics provided a great opportunity to benchmark the solution presented here to others; however, the methods behind these solutions have not yet been published in the scientific literature. For this reason, this method was analysed primarily by comparing it with baselines that could be evaluated under the same conditions.

The classical control theory baseline relies on information about the robot's relative location and orientation to the centreline of the lane, which is available in the simulator. This baseline works by controlling the robot to orient itself towards a point on its desired path ahead and calculating wheel velocities using a proportional-derivative (PD) controller based on the orientation error of the robot. The parameters of this controller are hand-tuned to achieve a sufficiently good performance, but more advanced control schemes could offer better results.

In many reinforcement learning problems (e.g. the Atari 2600 games [15]) the agents are compared to human baselines. Motivated by this benchmark, a method to measure how well humans were able to control Duckiebots was proposed, which was then used as a baseline. The values shown in Table 1 were recorded by controlling the simulated robot using the arrow keys on a keyboard (therefore via discrete actions), while the observations seen by the human driver were very similar to the observations of the reinforcement learning agent.

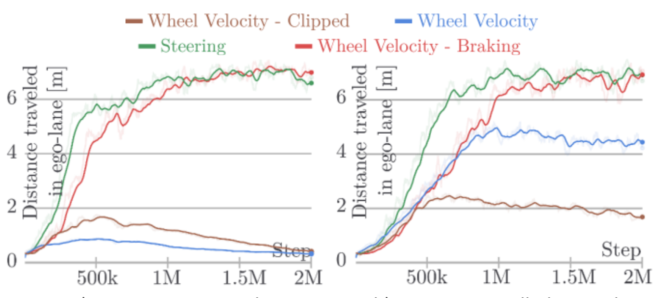
2.8. Methods to improve results at the AI Driving Olympics

The agents in this study were trained to solve autonomous driving problems in the Duckietown environment and not to maximise scores at the AI Driving Olympics. Therefore, some hyperparameters and methods had to be modified to match the competitions' evaluation procedures. It was found that training on lower frame rates (0.1 ms step time) improved the scores even though the evaluation simulation was stepped more frequently. In addition, implementing the same motion blur simulation that was applied in the official evaluation improved the results significantly compared with agents that were trained on non-blurred observations.

3. RESULTS

3.1. Simulation

Even though multiple papers have demonstrated the feasibility of training vision-based driving policies using reinforcement learning, adapting to a new environment still poses many challenges. Due to the high dimensionality of the image-like observations, many algorithms converge slowly and are very sensitive to hyperparameter selection. The method presented in this study, using proximal policy optimization, is able to converge with good lane-following policies in 1-million



a) Orientation reward b) Distance travelled reward

Figure 5. Learning curves for the reinforcement learning agent with different action representations and reward functions.

timesteps thanks to the high sample complexity of the algorithm. This training takes 2–2.5 hours on five cores of an Intel Xeon E5-2698 v4 2.2 GHz CPU and an Nvidia Tesla V100 GPU if 16 parallel environments are used.

3.1.1. Comparison against baselines

Table 1 compares the reinforcement learning agent from this study with the baselines. The performance of the trained policy is measurable to the classical control theory baseline as well as to how well humans are able to control the robot in the simulation. Most metrics indicate similarly good or equal performance even though the PD-controller baseline relies on high-level data such as position and orientation error rather than images.

3.1.2. Comparison against other solutions at the AI Driving Olympics

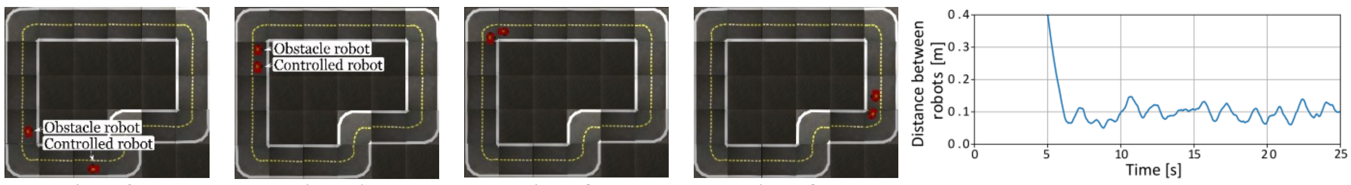
Table 2 shows the top-ranking solutions of the simulated lane-following (validation) challenge at the 5th AI Driving Olympics. All top-performing solutions were able to control the robot reliably in the simulation for the duration of an episode (60 s); however, the distances travelled were different. The method in this study is able to control the robot reliably at the highest speed, so it therefore achieves the highest distance-travelled value while also showing good lateral deviation and rarely departing from the ego-lane.

3.1.3. Action representation and reward shaping

Experiments with different action representations show that constrained and preferably biased action spaces allow convergence with good policies (*wheel velocity - braking* and *steering*). However, more general action spaces (*wheel velocity* and its *clipped* version) can only converge with inferior policies during the same number of steps (see Figure 5). The proposed orientation-based

Table 2. Comparing the method in this study with other solutions at the AI Driving Olympics

Author	t_{survive} in s ↑	s_{ego} in m ↑	d_d in m·s ↓	t_{out} in s ↓
A. Kalapos [10], [16]	60	**30.38**	2.65	**0**
A. Béres [16]	60	29.14	4.10	1.4
M. Tim [16]	60	28.52	3.45	0.4
A. Nikolskaya	60	24.80	3.15	1.6
R. Moni [16]	60	18.60	1.78	**0**
Z. Lorincz [16]	60	18.6	3.5	0.8
M. Sazanovich	60	16.12	4.35	3.4
R. Jean	60	15.5	3.28	**0**
Y. Belousov	60	14.88	5.41	9.8
M. Teng	60	11.78	2.92	**0**
P. Almási [7], [16]	60	11.16	**1.32**	**0**



a) $t = 0$ s Initial Positions

b) $t = 6$ s Catching up

c) $t = 8$ s

d) $t = 24$ s

Following the vehicle ahead

e) Approximate distance between the vehicles

Figure 6. Sequence of robot positions in a collision avoidance experiment with a policy trained using the modified *orientation* reward. After $t = 6$ s, the controlled robot follows the vehicle in front at a short but safe distance until the end of the episode (approximate distance is calculated as the distance between the centre points of the robots minus the length of a robot).

reward function also leads to as good a final performance as one that is 'trivially' rewarding based on the distance travelled; however, the latter seems to perform better on more general action representations (because policies using these action spaces and trained with the *orientation* reward do not learn to move fast).

3.2. Real-world driving

To measure the quality of the transfer learning process and the performance of the controller in the real world, performance metrics that were easily measurable both in reality and simulation were selected. These were recorded in both domains in matching experiments and compared against each other. The geometry of the tracks, the dimensions and the speed of the robot were simulated accurately to evaluate the robustness of the policy against all the inaccurately simulated effects and those that were not simulated. Using this method, policies trained in the domain-randomised simulation were tested as well as those that were trained only in the 'nominal' simulation. This allows for the evaluation of the transfer learning process and the highlighting of the effects of training with domain randomisation. The real and simulated version of the test track used in this analysis is shown in Figure 4 b and Figure 4 c.

During real evaluations, it was generally found that under ideal circumstances (no distracting objects at the side of the road and good lighting conditions), the policy trained in the 'nominal' simulation was able to drive reasonably well. However, training with domain randomisation led to a more reliable and robust performance in the real world.

Table 1 shows the quantitative results of this evaluation. The two policies seemed to perform equally well when compared based on their performance in the simulation. However, metrics recorded in the real environment show that the policy trained with domain randomisation performed almost as well as in the simulation, while the other policy performed noticeably worse. The lower *distance travelled ego-lane* metric of the domain-randomised policy can be explained by the fact that the vehicle tended to drift to the left-hand lane at sharp turns but returned to the right-hand lane afterwards, while the nominal policy usually made more serious mistakes. Note that in these experiments the *orientation*-based reward and the *steering* action representation were used, as this configuration learns to control the robot in the minimum number of steps and the shortest training time.

An online video demonstrates the performance of the trained agent from this study: https://youtu.be/kz7YWEmg1Is (Accessed 23 September 2021).

An important limitation for the method presented in this study is that during real evaluations, the speed of the robot had to be decreased to half of the simulated value. The policy evaluations were executed on a PC connected to the robot via wireless LAN; therefore, the observations and the actions were transmitted between the two devices at every step. This introduced delays in the order of $10 - 100$ ms, making the control loop unstable when the robot was moving at full speed. However, at half speed, a stable operation was achieved.

It was noticed that models trained with motion blur and longer step times for the AI Driving Olympics performed more reliably in the real world regardless of whether they used domain randomisation. However, further analysis and retraining of these agents multiple times is needed to firmly support these presumptions.

3.3. Collision avoidance

Figure 6 demonstrates the learned collision avoidance behaviour. In the first few seconds of the simulation, the robot controlled by the reinforcement learning policy accelerates to full speed. Then, as it approaches the slower, non-learning robot, it reduces its speed and maintains an approximately constant distance from the vehicle ahead (see Figure 6). From the simple, fully convolutional network of this policy, learning, planning and executing a more complex behaviour, such as overtaking, cannot be expected.

Table 4 shows that training with both reward functions leads to functional lane-following behaviour. However, the non-maximal *survival time* values indicate that neither of the policies are capable of performing lane following reliably with the presence of an obstacle robot for 60 s. All metrics in Table 4 indicate that the modified *orientation* reward leads to better lane-following metrics than the simpler *distance travelled* reward. It should be noted that these metrics were mainly selected to evaluate the lane-following capabilities of an agent; a more in-

Table 3. Evaluation results of reinforcement learning agent in the real environment and in matching simulations.

Eval. Domain	Mean metrics over 6 episodes		Domain Rand. Policy	Nominal Policy
Real	Survival time in s	↑	54	45
	Distance travelled both lanes in m	↑	15.6	11.4
	Distance travelled ego-lane in m	↑	7.0	8.4
Sim.	Survival time in s	↑	60	60
	Distance travelled in m	↑	15.5	15.0

Table 4. Evaluation results of policies trained for collision avoidance with different reward functions.

Mean metrics over 15 episodes		Distance travelled	Orientation $+r_{\mathrm{coll}}$
Survival time (max. 60) in s	↑	46	52
Distance travelled both lanes in m	↑	22.5	22.9
Distance travelled ego-lane in m	↑	22.7	23.1
Lateral deviation in m·s	↓	1.9	1.6
Orientation deviation in rad·s	↓	6.3	5.8

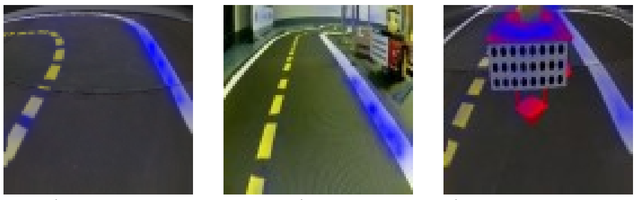


Figure 7. Salient objects highlighted on observations in different domains and tasks. Blue regions represent high activations throughout the network.

depth analysis of collision avoidance with a vehicle in front calls for more specific metrics.

An online video demonstrates the performance of the agent trained in this study: https://youtu.be/8GqAUvTY1po (Accessed 23 September 2021)

3.4. Salient object maps

Visualising which parts of the input image contribute the most to a particular output (action) is important because it provides some explanation of the network's inner workings. Figure 7 shows salient object maps in different scenarios generated using the method proposed in [17]. All of these images indicate high activations on lane markings, which is expected.

4. CONCLUSIONS

This work presented a solution to the problem of complex, vision-based lane following in the Duckietown environment using reinforcement learning to train an end-to-end steering policy capable of simulation-to-real transfer learning. It was found that the training is sensitive to problem formulation, such as the representation of actions. This study has demonstrated that by using domain randomisation, a moderately detailed and accurate simulation is sufficient for training end-to-end lane-following agents that operate in a real environment. The performance of these agents was evaluated by comparing some basic metrics to match real and simulated scenarios. Agents were also successfully trained to perform collision avoidance in addition to lane following. Finally, salient object visualisation was used to give an illustrative explanation of the inner workings of the policies in both the real and simulated domains.

ACKNOWLEDGEMENT

We would like to show our gratitude to Professor Bálint Gyires-Tóth (BME, Dept. of Telecommunications and Media Informatics) for his assistance and comments on the progress of our research.

The research reported in this paper and carried out at the Budapest University of Technology and Economics was supported by Continental Automotive Hungary Ltd. and the 'TKP2020, Institutional Excellence Programme' of the National Research Development and Innovation Office in the field of Artificial Intelligence (BME IE-MI-SC TKP2020).

REFERENCES

[1] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, Proc. Of the International Conference on Machine Learning, New York, United States, 19–24 June 2016, pp. 1928-1937.

[2] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, F. Nashashibi, End-to-end race driving with deep reinforcement learning, Proc. of the IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018, pp. 2070-2075.

[3] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, A. Shah, Learning to drive in a day, Proc. of the International Conference on Robotics and Automation (ICRA), Montreal, Canada, 20–24 May 2019, pp. 8248-8254.

[4] W. Shi, S. Song, Z. Wang, G. Huang, Self-supervised discovering of causal features: towards interpretable reinforcement learning, 2020. Online [Accessed 3 August 2020] https://arxiv.org/abs/2003.07069

[5] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, D. Karuppasamy, DeepRacer: educational autonomous racing platform for experimentation with Sim2Real reinforcement learning, 2019. Online [Accessed 13 April 2020] https://arxiv.org/abs/1911.01562

[6] M. Szemenyei, P. Reizinger, Attention-based curiosity in multi-agent reinforcement learning environments, Proc. of the International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), Majorca Island, Spain, 3–5 May 2019, pp. 176-181.

[7] P. Almási, R. Moni, B. Gyires-Tóth, Robust reinforcement learning-based autonomous driving agent for simulation and real world, Proc. of the International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 19–24 July 2020, pp. 1-8.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with deep reinforcement learning, 2013. Online [Accessed 13 April 2020] https://arxiv.org/abs/1312.5602

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017. Online [Accessed 2 December 2019] https://arxiv.org/abs/1707.06347

[10] A. Kalapos, C. Gór, R. Moni, I. Harmati, Sim-to-real reinforcement learning applied to end-to-end vehicle control, Proc. of the 23rd International Symposium on Measurement and Control in Robotics (ISMCR), Budapest, Hungary, 15–17 October 2020, pp. 1-6.

[11] J. Zilly, J. Tani, B. Considine, B. Mehta, A. F. Daniele, M. Diaz, G. Bernasconi, C. Ruch, J. Hakenberg, F. Golemo, A. K. Bowser, M. R. Walter, R. Hristov, S. Mallya, E. Frazzoli, A. Censi, L. Paull, The AI Driving Olympics at NeurIPS, 2018. Online [Accessed 13 April 2020] https://arxiv.org/abs/1903.02503

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, Proc. of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016, 14 pp. Online [Accessed 23 September 2021] http://arxiv.org/abs/1506.02438

[13] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, I. Stoica, Rllib: abstractions for distributed reinforcement learning, Proc. of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018, pp. 3053-3062.

[14] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, L. Paull, Duckietown environments for OpenAI gym, 2018. Online [Accessed 15 January 2021] https://github.com/duckietown/ gym-duckietown

[15] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (2013), pp. 253-279. DOI: 10.1613/jair.3912

[16] R. Moni, A. Kalapos, A. Béres, M. Tim, P. Almási, Z. Lőrincz, PIA project achievements at AIDO5, 2020. Online [Accessed 15 January 2021] https://medium.com/@SmartLabAI/pia-project-achievements-at-aido5-a441a24484ef

[17] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, U. Muller, Explaining how a deep neural network trained with end-to-end learning steers a car, 2017. Online [Accessed 15 April 2020] https://arxiv.org/abs/1704.07911

APPENDIX

Proximal policy optimization

The pseudo code for proximal policy optimization (PPO) is as follows:

Algorithm PPO, Actor-Critic Style (based on [9])

```
Input: initial policy with $\theta_0$ parameters and initial value function estimator with
  $\phi_0$ parameters
for iteration = 1,2,... do
    for actor=1,2,...,$N$ do
        Run $\pi_{\theta_{old}}$ in the environment for $T$ timesteps to collect $\tau_i$ trajectory
        Compute advantage estimates $\hat{A}_1, \dots, \hat{A}_T$ based on the current value
          function
    end
    Optimise $\mathfrak{L}_{CLIP}(\theta) + \mathfrak{L}_{KLPEN}(\theta)$ wrt. $\theta$, for K epochs and minibatch size
      $M \le NT$
    Fit the value function estimate by regression on mean-squared error
    $\theta_{old} \leftarrow \theta$, $\phi_{old} \leftarrow \phi$
end
```

The β adaptive parameter mentioned in Section 2.1 is updated according to the following rule:

$$\beta \leftarrow \begin{cases} \beta/2, \text{if } d < d_{\text{targ}}/1.5 \\ \beta \times 2, \text{if } d > d_{\text{targ}} \times 1.5, \end{cases} \tag{6}$$

where d_{targ} is a hyperparameter and d is the KL-divergence of the old and the updated policy

$$d = \hat{\mathbb{E}}\left[KL\left[\pi_{\theta_{\text{old}}}(\cdot\,|s_t), \pi_\theta(\cdot\,|s_t)\right]\right]. \tag{7}$$

The $\hat{A}_t$ generalised advantage estimate [12] is calculated as

$$\hat{A}_t = \sum_{l}^{\infty} (\gamma\lambda)^l \delta_t^V \tag{8}$$

$$\delta_t^V = r_t + \gamma\, V(s_{t+1}) - V(s_t)\,, \tag{9}$$

where $V(s_t)$ and $V(s_{t+1})$ are the value function estimates calculated by the value network at steps t and $t+1$; γ is the discount factor, while λ is a hyperparameter of the generalised advantage estimate.

To assure reproducibility, the hyperparameters of the algorithm are provided in the Table 5.

Table 5. Hyperparameters of the algorithm. The description of some parameters is from the RLlib documentation [13].

Description	Value
Number of parallel environments	$N = 16$
Learning rate	$\alpha = 5 \times 10^{-5}$
Discount factor for return calculation	$\gamma = 0.99$
λ parameter for the generalised advantage estimate	$\lambda = 0.95$
PPO clip parameter	$\epsilon = 0.2$
Sample batch size	$T = 256$
SGD minibatch size	$M = 128$
Number of epochs executed in every iteration	$K = 30$
Target KL-divergence for the calculation of β	$d_{\text{targ}} = 0.01$