~~Sim-to-real~~ Vision-based reinforcement learning ~~applied to end-to-end vehicle~~ for lane tracking control

András Kalapos*, Csaba Gór†, Róbert Moni‡ and István Harmati§

*BME, Dept. of Control Engineering and Information Technology, Budapest, Hungary, andras.kalapos.research@gmail.com

†Continental ADAS AI, Budapest, Hungary, csaba.gor@continental.com

‡BME, Dept. of Telecommunications and Media Informatics, Budapest, Hungary, robertmoni@tmit.bme.hu

§BME, Dept. of Control Engineering and Information Technology, Budapest, Hungary, harmati@iit.bme.hu

***Abstract*—In this work, we study vision-based end-to-end reinforcement learning on vehicle control problems, such as lane following and collision avoidance. Our controller policy is able to control a small-scale robot to follow the right-hand lane of a real two-lane road, while its training was solely carried out in a simulation. Our model, realized by a simple, convolutional network, only relies on images of a forward-facing monocular camera and generates continuous actions that directly control the vehicle. To train this policy we used Proximal Policy Optimization, and to achieve the generalization capability required for real performance we used domain randomization. We carried out thorough analysis of the trained policy, 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, we measured simple metrics on a real track and compared these with results from a matching simulation. Further analysis was carried out by visualizing salient object maps.**

***Index Terms*—artificial intelligence, autonomous vehicles, deep learning, Duckietown, machine learning, mobile robot, reinforcement learning, sim-to-real, transfer learning**

I. Introduction

Reinforcement learning has been used to solve many control and robotics tasks, however, only a handful of papers has been published so far that apply this technique to end-to-end driving [1]–[6]. Even fewer works focus on reinforcement learning-based driving, trained only in simulations but 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 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 generalization capability by testing on previously unseen tracks and on real diving videos, in an open-loop configuration, but their work didn't extend to evaluation on 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 involves training reinforcement learning agents in Duckietown similarly to ours, however, they mainly focused on presenting a method that explains the reasoning of the trained agents, rather than on the training methods. Similarly to our research, 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 the in-depth analysis of the road following policy. Almási et al. [7] used reinforcement learning to solve lane following in the Duckietown environment similarly to us. Their work differs from ours in using an off-policy reinforcement learning algorithm (Deep Q-Networks [8]), while we use an on-policy algorithm (Proximal Policy Optimization [12]), which achieves significantly better sample efficiency and shorter training times. Another important difference is that they apply a hand-crafted color threshold-based segmentation to the input images, whereas our method takes the "raw" images as inputs, which allows for more robust real performance.

(a) Simulated (b) Real (c) Collision avoidance

Fig. 1. Salient objects highlighted on observations in different domains and tasks. Blue regions represent high activations throughout the network.

~~In this contribution~~

This paper is an extended version of our original contribution [10]. It includes the results of the 5th AI Driving

† and ‡ contributed equally to this work

Olympics [11] and aims to improve on the description of our methods. In both works, we study vision-based end-to-end reinforcement learning on vehicle control problems and propose a solution that performs lane following in the real world, using continuous actions, without any real data provided by an expert (as in [3]). Also, we perform validation of the trained policies in both the real and simulated domains.

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

II. Methods

~~Salient objects highlighted on observations in different domains and tasks. Blue regions represent high activations throughout the network.~~ We trained a neural network-based controller that takes images from a forward-looking monocular camera and produces control signals to drive a vehicle in the right lane of a two-way road. The vehicle to be controlled is a small differential-wheeled mobile robot, a so-called 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 is to travel as far as possible under a given time, without leaving the road~~(while lane~~. Lane departure is allowed, but ~~not preferred).~~ it is not preferred. Despite the latest version of the Duckiebots being equipped with wheel encoders, our method is solely reliant on data from the robot's forward-facing monocular camera.

~~Training and evaluation code for this paper will be open sourced after the 5th AI-Driving Olympics and will be available onGitHub[2].~~

A. Reinforcement learning algorithm

In reinforcement learning, an agent interacts with the environment by taking a_t action, then the environment returns s_t observation and r_t reward. The agent computes the next action based on s_t and so on. The policy is the parametric controller of the agent that is tuned during the reinforcement learning training. Sequences of actions, observations, and rewards (so-called τ trajectories) are used to train the parameters of the policy to maximize 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, e.g., steering and throttle, and the observations are sensor data about the environment of the vehicle, e.g., camera, lidar data, or higher-level environment models. In this research, the observations are images from the robot's forward-facing camera, and the actions are velocity signals for the two wheels of the robot.

To train the policy we used Proximal Policy Optimization algorithm [12] for its stability, sample-complexity, and ability to take advantage of multiple parallel workers.

Policy optimization algorithms are on-policy reinforcement learning methods that ~~directly update the $\pi_\theta(a_t|s_t)$~~ optimize the parameters $\pi_\theta(a_t|s_t)$ policy based on the a_t actions and the r_t reward received for them~~(~~. θ denotes the trainable parameters of the policy~~and s_t is the observation at timestep t)~~ . ~~The policy used for these algorithms~~ On-policy reinforcement learning algorithms optimize the $\pi_\theta(a_t|s_t)$ policy based on trajectories in which the actions were computed by $\pi_\theta(a_t|s_t)$. In contrast to them, off-policy algorithms (such as DQN [8]) learn the estimate of the action-value function of the environment based on data collected by any agent, which makes these algorithms less stable in some environments. In policy optimization algorithms the $\pi_\theta(a_t|s_t)$ policy is stochastic and in case of deep reinforcement learning it ~~'s~~ is implemented by a neural network, which is updated using ~~gradient methods. In simpler versions of the algorithm (such as REINFORCE [?]) , the gradients are estimated by $\hat{g} = \hat{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) G^{\pi_\theta}(a_t, s_t)]$, where G_t is the return~~a gradient method. The policy is stochastic because, instead of computing the actions directly, the policy network predicts the parameters of a probability distribution (see μ and σ on fig. 2) that is sampled to acquire the $\tilde{a}_t$ predicted actions (here, predicted refers to this action being predicted by the policy).

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. [12]:

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

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

where $\text{clip}(\cdot)$ and $\text{KL}[\cdot]$ refer to the clipping function and KL-divergence respectively, while $\hat{A}$ is calculated as the generalized advantage estimate [13]. In these loss functions~~$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$~~, ϵ is a constant usually in the $[0.1, 0.3]$ range, while β is an adaptive parameter~~.~~, and

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

We used an open-source implementation of ~~the algorithm~~Proximal Policy Optimization from RLlib [14], which performs the gradient updates based on the weighted sum of these loss functions. Pseudo code and additional details for the algorithm are provided in the Appendix.

B. Policy architecture

The controller policy is realized by a shallow (4-layer) convolutional neural network. We consider this policy end-to-end because the only learning component is the neural network, which directly computes actions based on observations from the environment. Both the policy and the value network use the architecture presented by Mnih et al. [1], with ~~no weight sharing (with~~ the only difference of using linear activation on the output of the policy network~~).~~. No weights are shared between the policy and the value network.

[1] https://github.com/kaland313/Duckietown-RL

[2]

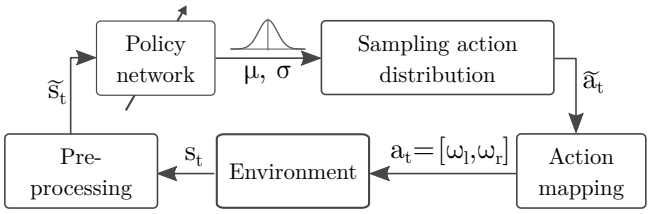


Fig. 2. Illustration of the policy architecture with the used notations. The agent or policy is represented by the "Policy Network" and "Sampling action distributions" blocks. s_t: "raw" observation, $\tilde{s}_t$: preprocessed observation, $\tilde{a}_t$: predicted action, a_t: postprocessed action

Some pre- and post-processing is applied to the observations and actions respectively, but these only perform very simple transformations ~~.~~ (explained in the next paragraph and sec. II-C). The aim of these pre- and postprocessing steps is to transform the s_t observations and a_t actions to representations that enable faster convergence without losing important features in the observations or restricting necessary actions.

The input of the policy network is the last three observations (images) scaled, cropped and stacked (along the depth axis). The observations returned by the environment (s_t on fig. 2) are 640×480 (width, height) RGB images whose top third mainly shows the sky, therefore is cropped. Then, the cropped images are scaled down to 84×84 resolution (note the uneven scaling), which are then stacked along the depth axis resulting in $84 \times 84 \times 9$ input tensors ~~.~~ ($\tilde{s}_t$ on fig. 2). The last three images are stacked to provide the policy with information about the robot's speed and acceleration.

We experimented with multiple action representations (see sec. II-C)~~, depending on these~~. Depending on these, the policy outputs ~~one or two scalar values which control~~ $\tilde{a}_t$ predicted action vector of two or a scalar value that controls the vehicle. The policy is stochastic~~, therefore~~; therefore, the output of the neural network produces the μ and $\log\sigma$ parameters of a ~~(multivariate diagonal) normal distribution , which~~ multivariate diagonal normal distribution During training, this distribution is sampled to acquire ~~actions.~~ the $\tilde{a}_t$ action, which improves the exploration of the action space. During evaluations, the sampling step is skipped by using the predicted μ mean value as the $\tilde{a}_t$ policy output.

C. Action representations

The action post-processing step transforms the $\tilde{a}_t$ predicted actions, which can be implemented using many representations, to $a_t = [\omega_l, \omega_r]$ wheel velocities (see fig 2). The vehicle to be controlled is a differential-wheeled robot, therefore the most general action representation is to directly ~~predict~~ compute the angular velocities of the two wheels as continuous values in the $\omega_{l,r} \in [-1; 1]$ range (where 1 and -1 correspond to rotating forward and backward at full speed). However, this action space allows for actions that are not necessary for the maneuvers we examine in this paper. Moreover, by allowing unnecessary actions, the reinforcement learning algorithm must rule these out, potentially making the exploration of the action space more difficult therefore increasing the steps required to train an agent. Several methods can be used to constrain and simplify the action space, such as discretization, clipping some actions, or mapping to a lower-dimensional space.

Most previous works (~~[1], [2], [5]~~ [1], [2], [5], [7]) use discrete action spaces, thus the neural network in these policies selects one from a set of hand-crafted actions (steering, throttle combinations), while Kendall et al. [3] utilize continuous actions, as we do. ~~However, they don't predict throttle directly, only a speed set-point for a classical controller.~~

In order to test the reinforcement learning algorithm's ability to solve the most general problem, we experimented with multiple action mappings and simplifications of the action space. These were

1) Wheel Velocity: The policy directly outputs wheel velocities, $a_t = [\omega_l, \omega_r] = \tilde{a}_t$, therefore $\omega_{l,r} \in [-1; 1]$

2) Wheel Velocity - Positive Only: Only allow positive wheel velocities, because only these are required to move forward. Values predicted outside the $\omega_{l,r} \in [0; 1]$ interval are clipped. $a_t = [\omega_l, \omega_r] = \text{clip}(\tilde{a}_t, 0, 1)$

3) Wheel Velocity - Braking: Wheel velocities could still only fall in the $\omega_{l,r} \in [0; 1]$ interval, but the predicted values are interpreted as the amount of braking from the maximum speed~~$\omega_{l,r} = 1 - y_{pred,l,r}$~~. The main differentiating factor from the Positive Only option is the bias towards moving forward at full speed. $a_t = [\omega_l, \omega_r] = \text{clip}(1 - \tilde{a}_t, 0, 1)$

4) Steering: Predicting a scalar value that is continuously mapped to combinations of wheel velocities. The 0.0 scalar value corresponds to going straight (at full speed), while -1.0 and 1.0 refer to turning left or right, with one wheel completely stopped and the other one 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. A formula that implements this action mapping: $[\omega_l, \omega_r] = \text{clip}([1 + \tilde{a}_t, 1 - \tilde{a}_t], 0, 1)$

D. 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 optimization problem. The primary objective for the agent is to travel as far as possible under a given time in the right lane, therefore we propose two rewards that promote this behavior.

1) Distance traveled: The agent is directly rewarded proportionally to the distance it moved further along the right lane under every step. Only longitudinal motion is counted, and only if the robot stayed in the right lane.

2) Orientation: The agent is rewarded if it is facing towards and moves in a certain desired orientation, which is determined based on its lateral position. In simple terms, it is rewarded the most if it faces towards the center of the right lane (some example configurations are shown on fig. 3d). A term

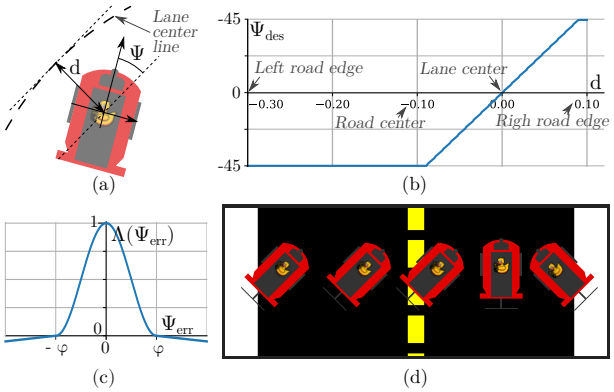


Fig. 3. Explanation of the proposed Orientation reward. (a) explains Ψ, d, (b) shows how the desired orientation depends on the lateral error, (~~d~~c) shows the function $\Lambda(x)$, while (d) shows some examples of desired configurations~~, while (c) shows the function~~.

$$\Lambda(x) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\left(\pi \frac{x}{\varphi}\right) & \text{if } -1 \geq x \geq 1 \\ \varepsilon(-|\frac{x}{\varphi}| + 1) & \text{otherwise} \end{cases}, \varepsilon \in [10^{-1}, 10^{-2}] \quad (3)$$

proportional to the angular velocity of the faster moving wheel is also added to encourage fast motion.

This reward is 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~~ orientation- and velocity-based components (explained below), while λ_Ψ, λ_v constants scale these to $[-1, 1]$. Ψ, d are orientation and lateral error from the desired trajectory, which is the center line of the right lane (see fig. 3a).

The orientation-based term is calculated as $r_\Psi(\Psi, d) = \Lambda(\Psi_{err}) = \Lambda(\Psi - \Psi_{des}(d))$, where $\Psi_{des}(d)$ is the desired orientation, calculated based on the lateral distance from the desired trajectory (see fig. 3b for the illustration of $\Psi_{des}(d)$). The Λ function achieves that $|\Psi_{err}| < \varphi$ error is promoted largely, while error larger than this leads to small negative reward (~~formal description and~~ a plot of Λ is shown on fig. 3c).

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

where $\varepsilon \in [10^{-1}, 10^{-2}]$ and $\varphi = 50°$ hyper-parameter ~~was~~ were selected arbitrarily.

The velocity-based component is calculated as $r_v(\omega_l, \omega_r) = \max(\omega_l, \omega_r)$ to reward high speed motion equally in straight and curved sections~~, where~~. In curves, only the outer wheel can rotate ~~as fast as on straight sections~~at maximal speed, on a straight road, both of them.

E. Simulation to reality transfer

To train agents, we used an open-source simulation of the Duckietown environment [15]. It models certain physical properties of the real environment accurately (dimensions of the robot, camera parameters, dynamic properties etc.), but several other effects (textures, objects surrounding the roads) and light simulation are less realistic (e.g. compared to modern computer games). These inaccuracies create a gap between simulation and reality which makes it challenging for any reinforcement learning agent to be trained in a simulation but operate in reality.

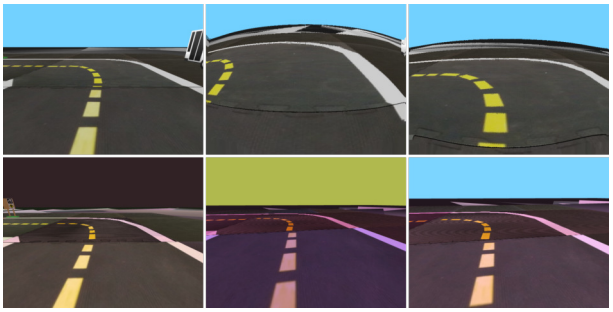


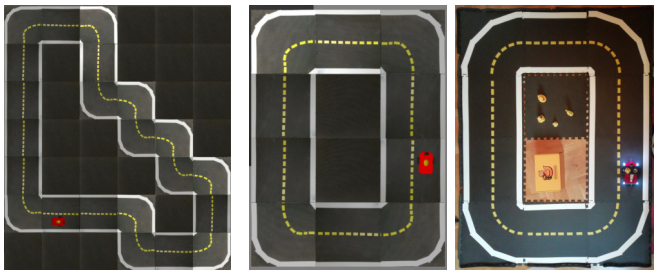
Fig. 4. Examples of domain randomized observations

To bridge the simulation to reality gap, and to achieve the generalization capability required for real performance we used domain randomization. This involves training the policy in many different variants of a simulated environment, by varying lighting conditions, object textures, camera, and vehicle dynamics parameters, road structures etc. (for examples of domain randomized observations see fig. 4). In addition to the "built-in" randomization options of Gym-Duckietown, we trained on a diverse set of maps to further improve the agent's generalization capability.

F. Collision avoidance

Collision avoidance with other vehicles greatly increases the complexity of the lane-following task. These problems can be solved in different ways, e.g. by overtaking or following from 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 160° horizontal field of view, therefore the policy controlling the vehicle has no information about objects moving next to or behind the robot. Also, for simplicity, we ~~chose a convolutional network and didn't incorporate~~ used the same convolutional network for collision avoidance as for lane following, which does not feature an LSTM cell ~~into it~~or any other sequence modelling component (in contrast to [2]). For these reasons, it is unable to plan long maneuvers, such as overtaking, which also requires side-vision to check if returning to the right lane is safe. Therefore, we trained a policy in situations where there is a slow vehicle ahead, and the agent has to learn to perform lane following at full speed until it catches up with the vehicle upfront, then it must reduce its speed and keep a safe distance to avoid collision.

In these experiments, the *Wheel Velocity - Braking* action ~~mapping was used~~ representation was used as the policy's output because this allows the ~~policy~~ agent to slow down or even stop the vehicle if necessary (unlike the one we call *Steering*). ~~Rewards~~ Both the *Orientation* and the *Distance traveled*



(a) Simulated (b) Simulated (c) Real

Fig. 5. (a): Test track used for simulated reinforcement learning and baseline evaluations. (b),(c): Real and simulated test track used for the evaluation of the simulation to reality transfer.

reward functions were used to train agents for collision avoidance~~were the modified version of the Orientation reward and Distance traveled (unchanged)~~. The former one was supplemented with a term that promotes collision avoidance, the latter is used unchanged. The simulation we used provides a ~~p_{coll}~~ p_{coll} penalty if the so-called safety circles of two vehicles overlap. The ~~reward term~~ r_{coll} reward component that promotes collision avoidance is calculated based on this penalty. If the penalty is decreasing, which is because the robot is able to increase the distance from an obstacle, the reward term is proportional to the change of the penalty. Otherwise, it is ~~proportional to its change if it's decreasing, otherwise, it 's~~ 0.

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

This term is added to the Orientation reward and intends to encourage the policy to increase the distance from the vehicle ahead if it got too close. Collisions are only penalized by terminating the episode, without giving any negative reward.

G. Evaluation

To assess the performance of the reinforcement learning-based controller, we measured multiple performance metrics in the simulation and compared these against two baselines, one using a classical control theory approach, and human driving. ~~To our knowledge no other methods have been published so far, which could be used as a baseline.~~ These metrics are:

1) Survival time ($t_{survive}$) [s]: The time until the robot left the road or the time period of an evaluation ~~.~~episode.

2) Distance traveled in ego-lane (s_{ego}) [m]: The distance traveled along the right-hand-side lane under a fixed time period. Only longitudinal motion is counted, therefore tangential movement counts the most towards this metric.

3) Distance traveled both lanes (s_{both}) [m]: The distance traveled along the road under a fixed time period, but also sections where the agent moved in the oncoming lane count towards this metric.

4) Lateral deviation (d_d) [m·s]: Lateral deviation from the lane center line integrated over the time of an episode.

5) Orientation deviation (d_Ψ) [rad·s]: The robot orientation's deviation from the tangent of the lane center line, integrated over the time of an episode.

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

Even thorough Duckietown intends to be a standardized 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 provides a great way to benchmark our solution to others; however, the methods behind these solutions are not published in the scientific literature. For this reason, we primarily analyze our method by comparing it to baselines that we can evaluate under the same conditions.

The classical control theory baseline relies on information about the robot's relative location and orientation to the centerline of the lane, which are available in the simulator. This baseline works by controlling the robot to orient itself towards a point on it's 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 were hand-tuned to achieve sufficiently good performance, but more advanced control schemes could offer better results.

In many reinforcement learning problems (e.g. the Atari 2600 games [16]) the agents are compared to human baselines. Motivated by this benchmark we propose a method to measure how well humans are able to control duckiebots, which could be used as a baseline. The values shown in Table I 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 compared to the observations of the reinforcement learning agent.

H. Methods/Tweaks/Tricks to improve results at the AI Driving Olympics

We trained our agents to generally solve autonomous driving problems in the Duckietown environment and not to maximize scores at the AI Driving Olympics. Therefore, some hyperparameters and methods had to be modified to match the competitions' evaluation procedures. We found that training on lower frame rates (0.1 ms step time) improves the scores, even though the evaluation simulation is stepped more frequently. Also, implementing the same motion blur simulation that is applied in the official evaluation improves the results greatly over agents that were trained on non-blurred observations.

III. Results

A. Simulation

Even though multiple papers demonstrate 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. Our method, using

TABLE I
COMPARISON OF THE REINFORCEMENT LEARNING AGENT TO TWO BASELINES IN SIMULATION

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

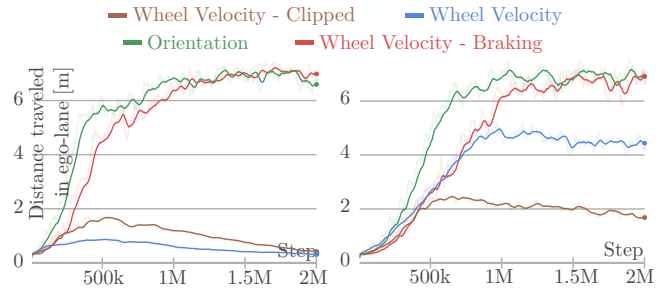
TABLE II
COMPARING OUR METHOD TO OTHER SOLUTIONS AT THE AI DRIVING OLYMPICS

Author	$t_{survive}$ [s] ↑	s_{ego} [m] ↑	d_d [m·s] ↓	t_{out} [s] ↓
A. Kalapos (ours, [10], [17])	60	**30.38**	2.65	**0**
A. Béres [17]	60	29.14	4.10	1.4
M. Tim [17]	60	28.52	3.45	0.4
A. Nikolskaya	60	24.80	3.15	1.6
R. Moni [17]	60	18.60	1.78	**0**
Z. Lorincz [17]	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], [17]	60	11.16	**1.32**	**0**

Datasource:`https://challenges.duckietown.org/v4/`
Downloaded at January 12, 2021

Proximal Policy Optimization is able to converge to good lane following policies in 1 million timesteps, thanks to the high sample-complexity of the algorithm.This training takes 2-2.5 hours on 5 cores of an Intel Xeon E5-2698 v4 2.2 GHz CPU and an Nvidia Tesla V100 GPU if 16 parallel environments are used.

1) Comparing against baselines: Table I compares our reinforcement learning agent to the baselines. The performance of the trained policy is measurable to our classical control theory baseline, as well as to how well humans are able to control the robot in the simulation. Most metrics indicate



(a) Orientation reward (b) Distance travelled reward

Fig. 6. Learning curves for the reinforcement learning agent with different action representations and reward functions.

TABLE III
EVALUATION RESULTS OF REINFORCEMENT LEARNING AGENT IN THE REAL ENVIRONMENT AND IN MATCHING SIMULATIONS

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

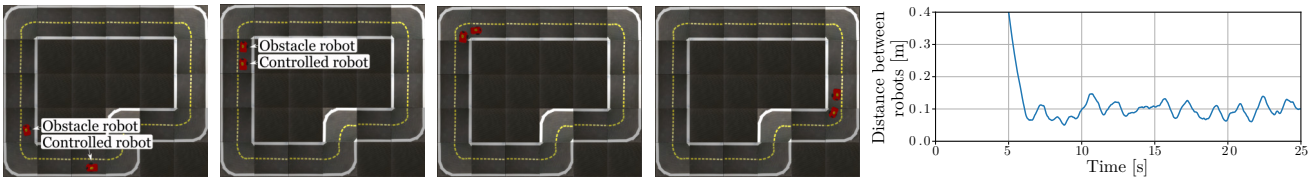
similarly good or equal performance, even though the PD controller baseline relies on high-level data such as position and orientation error, rather than images.

2) Comparing against other solutions at the AI Driving Olympics: Table II shows the top-ranking solutions of the simulated lane following (validation) challenge at the 5th AI Driving Olympics. All top-performing solutions are able to control the robot reliably in the simulation for the time of an episode (60 s), however, the distances traveled are different. Our method is able to control the robot reliably at the highest speed therefore achieves the highest distance traveled value, while also showing good lateral deviation and rarely departing the ego-lane.

3) Action representation and reward shaping: Experiments with different action representations show that constrained and preferably biased action spaces allow convergence to good policies (*Wheel Velocity - Braking* and *Steering*), however, more general action spaces (*Wheel Velocity* and ~~it's~~ its *Clipped* version) can only converge to inferior policies under the same number of steps (see fig. 6). The proposed orientation based reward function also leads to as good final performance as "trivially" rewarding based on the distance traveled, however, the latter seems to perform better on more general action representations (because policies using these action spaces and trained with the Orientation reward ~~doesn't~~ does not learn to move fast).

B. Real-world driving

To measure the quality of the transfer learning process and the performance of the controller in the real world, we selected performance metrics that are easily measurable both in reality and simulation. These were recorded in both domains in matching experiments and compared against each other. The geometry of the tracks, the dimensions, and speed of the robot are simulated accurately enough, to evaluate the robustness of the policy against all inaccurately and not simulated effects. Using this method, we tested policies trained in the domain randomized simulation, but also ones that were trained only in the "nominal" simulation. This allows us to evaluate the transfer learning process and highlight the effects of training with domain randomization. The real and simulated version of the test track used in this analysis is shown on fig. 5b and 5c.



(a) $t = 0$[s] Initial positions (b) $t = 6$[s] Catching up (c) $t = 8$[s] (d) $t = 24$[s] (e) Approximate distance between the vehicles.

Fig. 7. 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 of it from a short, but safe distance until the end of the episode. (Approximate distance is calculated as the distance between the center points of the robots minus the length of a robot.)

During real evaluations, generally, we experienced that under ideal circumstances (no distracting objects outside the roads and good lighting conditions) the policy trained in the "nominal" simulation is able to drive reasonably well. However, training with domain randomization leads to more reliable robust performance in the real world.

Table III show the quantitative results of this evaluation. The two policies seem to perform equally well if comparing them based on their performance in the simulation. However, metrics recorded in the real environment show that the policy trained with domain randomization performs almost as well as in the simulation, while the other policy performs noticeably worse. The lower *Distance traveled ego-lane* metric of the domain randomized policy is because the vehicle tends to drift to the left lane in sharp turns but returns to the right-lane afterward, 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 least amount of steps and training time.

An online video demonstrates the performance of our trained agent: `https://youtu.be/kz7YWEmg1Is`

An important limitation of our method is that during real evaluations, we had to decrease the speed of the robot to half of the simulated value. The policy evaluations are executed on a PC connected to the robot via wireless LAN; therefore, the observations and the actions are transmitted between the two devices at every step. This introduces delays in the order of 10-100 ms, which makes the control loop unstable if the robot is moving at full speed. However, at half speed, stable operation can be achieved.

We noticed that models trained with motion blur and longer step times for the AI Driving Olympics perform more reliably in the real world, regardless of using domain randomization. However, further analysis and retraining of these agents multiple times is needed to firmly support these presumptions.

C. Collision avoidance

Fig 7 demonstrates the learned collision avoidance behavior. 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 ~~it's~~ its speed and maintains approximately a constant distance from the vehicle ahead (see fig 7e). From the simple, fully-convolutional network of our policy, learning, planning, and executing a more complex behavior, such as overtaking, can not be expected.

TABLE IV
EVALUATION RESULTS OF POLICIES TRAINED FOR COLLISION AVOIDANCE WITH DIFFERENT REWARD FUNCTIONS

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

Table IV shows that training with both reward functions lead to functional lane-following behavior, 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 seconds. All metrics in Table IV indicate that the modified Orientation reward leads better lane following metrics, than the simpler Distance traveled reward. It should be noted, that these metrics were mainly selected to evaluate the lane following capabilities of an agent, more in-depth analysis of collision avoidance with a vehicle upfront call for more specific metrics.

An online video demonstrates the performance of our trained agent: `https://youtu.be/8GqAUvTY1po`

D. Salient object maps

Visualizing 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. Fig. 1 shows salient object maps in different scenarios, generated using the method proposed in [18]. All of these images indicate high activations on lane markings, which is expected.

IV. 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. We found that the training is sensitive to problem formulation, for example to the representation of actions. We showed that by using domain randomization, 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 in matching real and simulated scenarios. Agents were also successfully trained to perform collision avoidance in addition to lane following. Finally, salient object visualization was used to give an illustrative explanation of the inner workings of the policies, in both the real and simulated domains.

Acknowledgment

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 Program" 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, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[2] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075, 2018.

[3] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254, 2019.

[4] W. Shi, S. Song, Z. Wang, and G. Huang, "Self-supervised discovering of causal features: Towards interpretable reinforcement learning," *arXiv e-prints arXiv:2003.07069*, 2020.

[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, and D. Karuppasamy, "DeepRacer: Educational Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning," *arXiv e-prints arXiv:1911.01562*, Nov. 2019.

[6] M. Szemenyei and P. Reizinger, "Attention-based curiosity in multi-agent reinforcement learning environments," in *International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*, pp. 176–181, IEEE, 2019.

[7] P. Almási, R. Moni, and B. Gyires-Tóth, "Robust reinforcement learning-based autonomous driving agent for simulation and real world," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2020.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning."

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, vol. abs/1707.06347, 2017.

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

[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, and L. Paull, "The ai driving olympics at neurips 2018," *arXiv preprint arXiv:1903.02503*, 2019.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint, vol. abs/1707.06347, 2017.
R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[14] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*, pp. 3053–3062, 2018.

[15] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for openai gym," 2018.

[16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[17] R. Moni, A. Kalapos, A. Béres, M. Tim, P. Almási, and Z. Lőrincz, "Pia project achievements at aido5," 2020.

[18] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint*, vol. abs/1704.07911, 2017.

APPENDIX

Proximal Policy Optimization

The pseudo code for Proximal Policy Optimization is as follows:

Algorithm 1: Algorithm PPO, Actor-Critic Style (based on [12])

Input: initial policy, with θ_0 parameters and initial value function estimator with ϕ_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 τ_i trajectory
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ based on the current value function
  end
  Optimize $\mathfrak{L}_{CLIP}(\theta) + \mathfrak{L}_{KLPEN}(\theta)$ wrt. θ, for K epochs and minibatch size $M \leq NT$
  Fit the value function estimate by regression on mean-squared error
  $\theta_{old} \leftarrow \theta$, $\phi_{old} \leftarrow \phi$
end

The β adaptive parameter metnioned in section II-A is updated according tot he following rule:

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

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

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

$\hat{A}_t$ generalized advantage estimate [13] 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 generalized advantage estimate.

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

TABLE V
HYPERPARAMETERS OF THE ALGORITHM. THE DESCRIPTION OF SOME PARAMETERS ARE FROM THE DOCUMENTATION OF RLLIB [14]

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	$d_{targ} = 0.01$