2017 IEEE Intelligent Vehicles Symposium (IV)
June 11-14, 2017, Redondo Beach, CA, USA

Learning How to Drive in a Real World Simulation with Deep Q-Networks

Peter Wolf[1], Christian Hubschneider[1], Michael Weber[1],
André Bauer[2], Jonathan Härtl[2], Fabian Dürr[2],
J. Marius Zöllner[1,2]

***Abstract*— We present a reinforcement learning approach using Deep Q-Networks to steer a vehicle in a 3D physics simulation. Relying solely on camera image input the approach directly learns steering the vehicle in an end-to-end manner. The system is able to learn human driving behavior without the need of any labeled training data.**

An action-based reward function is proposed, which is motivated by a potential use in real world reinforcement learning scenarios. Compared to a naïve distance-based reward function, it improves the overall driving behavior of the vehicle agent. The agent is even able to reach comparable to human driving performance on a previously unseen track in our simulation environment.

I. INTRODUCTION

Ever since ALVINN [1] the dream has been to steer an autonomous car by using a single camera installed in the front of the vehicle. Building on this idea, we developed a system capable of learning how to drive autonomously in a real world simulation.

Many modern vehicles are already equipped with advanced driver assistance systems that are the result of current research, such as traffic sign detection, lane keeping or adaptive cruise control. In continuation of this trend, various car manufacturers and technology companies are working towards self-driving vehicles.

To drive autonomously, it is necessary for the vehicle to have knowledge of its surrounding environment and to adapt to it. This is achieved by most of today's autonomous driving systems through the employment of mediated perception approaches. Those treat the whole system as a combination of multiple components, each responsible for a specific task. The results provided by the components are then fused to obtain a model of the vehicle's environment, which provides the control system with all relevant information to plan driving behavior.

Deep learning has proven to be a powerful tool to solve Computer Vision tasks, especially Convolutional Neural Networks (ConvNets). ConvNets are able to learn a high degree of abstraction given raw input data by extracting features of different granularity. They have been successfully used to implement various components for autonomous systems.

[1] FZI Research Center for Information Technology, 76131 Karlsruhe, Germany. wolf, hubschneider, michael.weber, zoellner@fzi.de

[2] Karlsruhe Institute of Technology (KIT), Germany. andre.bauer, jonathan.haertl,fabian.duerr2@student.kit.edu

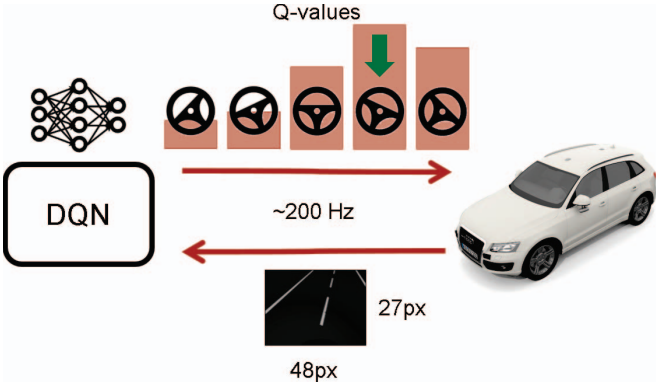


Fig. 1: Architecture of our system: Given visual input from the simulation, the DQN calculates Q-value estimates for steering actions. The steering action with the highest Q-value is chosen and executed by the simulated vehicle.

Recent work has further trained ConvNets with supervised learning for end-to-end systems [2]. However, to train those ConvNets sufficiently, a huge amount of labeled data of human driving is necessary.

We introduce an approach to train an autonomous vehicle to drive in a 3D real world physics simulation using deep reinforcement learning. Reinforcement learning enables the vehicle agent to learn from its own behavior instead of labeled data. Using a real world simulation with software components also employed in current real autonomous vehicles is a step towards driving on real streets. A schematic of our approach is depicted in Fig. 1.

The vehicle's main task is to stay in its designated lane using only visual information as input. We examined various reward components based on the current state of the vehicle aiming to learn the desired driving behavior.

The remainder of this paper is structured as follows: In Section II we give an overview about common approaches to the task of autonomous driving, as well as recent reinforcement learning methods. Section III introduces our approach describing the used architecture. This includes a detailed description of our reward concept. The evaluation of our approach is presented in Section IV, before we discuss our results in Section V.

978-1-5090-4804-5/17/$31.00 ©2017 IEEE

II. RELATED WORK

Many current autonomous driving systems use a *mediated perception* approach in their architecture [3, 4]. Mediated in this context means the perception system consists of various components that are solving just one dedicated part of the whole task. For example, there are components on the subject of the visual detection of lane markings [5, 6], cars [7, 8] , pedestrians [9, 10], traffic signs [11, 12], traffic lights [13, 14] or combinations of the aforementioned [15, 16]. Other non-camera sensors, like LiDAR or radar, are also used in some systems for obstacle detection [17, 18].

The outputs of these low level components are usually processed by a fusion component to generate a representation of the vehicle's environment [3, 19]. This environment model is used by a further component to plan and control the vehicle's behavior.

In [4] a *direct perception* approach is proposed. Instead of using detections of lane markings and other objects to derive requirements for the vehicle's behavior indirectly, a set of affordances for driving actions is defined. Those affordances include the heading of the vehicle, the distances to surrounding lane markings and preceding cars. A ConvNet is trained to extract the defined affordances directly from the visual sensor input. Steering commands and also the decision to overtake preceding cars are issued by a simple controller based on the affordances' values. While *direct perception* reduces the complexity of the environment model, the affordances and the vehicle's controller are hand-crafted.

Going one step further, the sensory input can be directly mapped to a control command using a neural network. This approach first came up in [1, 20]. Very recently NVIDIA [2] was able to successfully train a ConvNet to map raw camera images to steering angles. The end-to-end system is able to drive on highways and even roads without lane markings.

In contrast to this supervised learning approach, there have also been publications focusing on end-to-end reinforcement learning that do not require pre-recorded driving data. To drive solely based on the visual input in the open racing simulation TORCS, in [21] and [22] Koutník et al. train recurrent neural networks and ConvNets using an evolutionary reinforcement learning approach. The networks are used to perceive the car's environment and to control the car and are evolved based on a fitness function. Mnih et al. let their system learn to play a wide range of Atari 2600 games on or above human level with their proposed deep Q-network (DQN) [23, 24]. The DQN has powerful generalizability and since its inception, various extensions have been published, ranging from a continuous action space [25], asynchronous agents [26] and prioritizing experience replay [27] to extended network architectures for reducing overestimation [28] and to separate state values and action advantages [29]. While most of these DQN approaches are evaluated on an Atari 2600 benchmark [30], [25] and [26] also successfully train the DQN to drive a vehicle in TORCS.

Our approach also makes use of a DQN to train a controller to drive a vehicle. In contrast to these approaches, our controller learns to drive like a human traffic participant, not like an arcade racing driver. This means a smooth driving within the designated lane. For this purpose, our simulated vehicle is modeled in accordance with an Ackermann steering model and behaves more closely to a real vehicle.

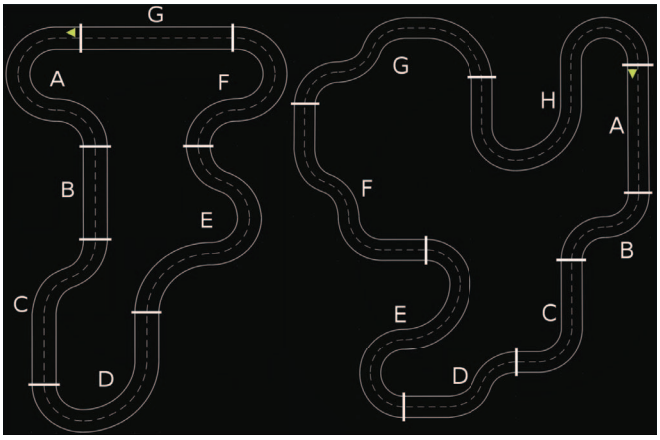


Fig. 2: Tracks used for training (left) and additional evaluation (right) with separate sections named for reference. The green triangle marks the cars starting position for the evaluation rounds.

III. APPROACH

A. Framework

For our simulation we use the simulation environment proposed in [31], which extends the realistic and efficient physics simulation Gazebo [32]. This simulation environment allows us to build tracks (see Fig. 2) and generate corresponding lanelet maps [33], which are a basis for the agent's reward and provide a ground truth for the evaluation. Each track consists of two lanes which are 2.75m wide and are separated by a dashed center line. The vehicle model provided by the simulation environment is based on the Ackermann steering model. The deep learning framework Caffe [34] is used to train the convolutional architecture of the DQN. The components of our framework are implemented as independent nodes within the Robot Operating System (ROS) [35]. Using ROS as middleware enables an effortless exchange of data and control commands between the simulation environment and the DQN. In order to provide an image of the vehicle's environment to the DQN, a camera with a resolution of 48×27 pixel is mounted in the front of the car. It is facing forward to capture the lane markings lying ahead. Furthermore, the simulation yields the vehicle's position and heading, on which the reward is determined. To control the car, we define a set of 5 actions, which set the vehicle's steering angle to specified values while driving with a constant velocity. The actions and their respective steering angles are shown in Tab. I.

In contrast to the Arcade Learning Environment [30] used to train and evaluate agents playing Atari games, the simulation used in this paper does not wait for the input from our trained agent, but is constantly running. This means the

Left	Half-Left	Straight	Half-Right	Right
-0.5	-0.25	0.0	0.25	0.5

TABLE I: The set of actions (steering angle in rad) used by the DQN to control the vehicle.

vehicle is moving with the steering angles set by the last chosen action until a new angle is set. Due to this, it is necessary for the agent to have a very fast response time. If the time that passes in the simulation between sending the camera image and receiving the control command, the chosen action might not be appropriate anymore.

B. Network

Our system makes use of the DQN architecture as proposed by Mnih et al. [23]. DQN has been shown to be able to outperform humans in various Atari games. Using the visual information of the screen as input and the game's intrinsic reward, the DQN agent learns the Q-values of the individual actions for the input image. To deal with common instabilities of reinforcement learning when dealing with high-dimensional inputs, a technique called *experience replay* [36] is utilized to decorrelate the observations and to smooth out the distribution of the training data. Instead of using the observation in sequence, the network's observations are saved in memory and randomly sampled when updating its weights.

In contrast to the Atari playing DQN, a 48×27 pixel gray scale input image is used. At this reduced resolution images still contain sufficient information about the lane markings ahead of the vehicle. To accommodate the new input dimensions, both convolutional layers apply 4×4 kernels with a stride of 2. The size reduction is necessary as the original kernel size of 8 for the first convolutional layer reduces the size of the input data too much. Additionally, the input consists of only a single image instead of the last 4 frames, as an appropriate steering angle can be determined without knowledge of past states. This reduction of necessary input data increases the frame-rate of the system as the amount of data transferred between nodes is a bottleneck within the system.

C. Training

We trained the DQN by letting the vehicle agent drive on a training course (see Fig. 2) in the simulation environment. Throughout the training of the DQN an action has to be selected for every given input image, as we do not employ frame-skipping. For exploration purposes, an ϵ-greedy policy is used with ϵ giving the probability for selecting an action randomly. Otherwise the DQN estimates the Q-values for every action given the input image. The action with the maximum Q-value is selected. At the start of the training ϵ is set to 1, but it is linearly reduced with an increasing number of iterations until a minimum value of 0.1 is reached after 5 million iterations.

During every training step, the given input, the chosen action, the resulting state and the immediate reward are saved as a transition in the replay memory. Once the size of the memory has reached the minimum required size of 5000 transitions, 32 randomly sampled transitions are used to update the weights of the DQN after every step. When replay memory's maximum capacity of 500k transitions is reached, an incoming transition replaces the oldest stored transition. During the weight update the simulation environment is paused, since in this phase the DQN is not able to steer the vehicle.

The training is done in episodes, where each episode marks a separate run of the vehicle through the training course. An episode is terminated, if the vehicle's distance to the center line of its lane surpasses a defined threshold ρ. If not specified otherwise, we use $\rho = 4.125$ m, which corresponds to 1.5 times the lane width. Additionally, to guarantee periodic terminal rewards, an episode is run for a maximum of 10k iterations or until a reward of 10k is reached. After an episode finishes, the vehicle is reset on a random part of the course to start the next episode.

We trained our DQN using three different gradient-based optimization methods to examine impacts on the training of our DQN: a stochastic gradient descent (SGD), Adam and RMSProp algorithm. Each of them was used to train the network for 2 million iterations. For the SGD a learn rate of 10^{-5} was chosen, while the Adam and the RMSProp

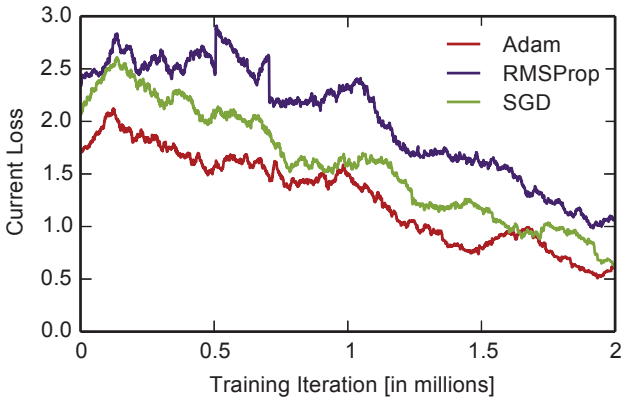


Fig. 3: Smoothed Loss per training iteration.

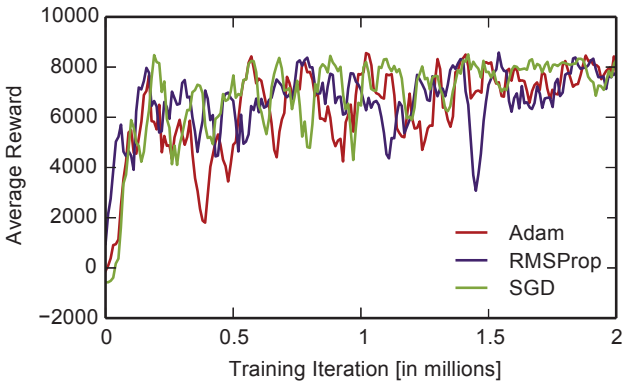


Fig. 4: Smoothed average reward per training iteration.

solvers used a learn rate of 10^{-4}. Furthermore, for Adam $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon_{Adam} = 10^{-8}$ was used. The decay for the RMSProp was set to 0.98. The development of the net loss for the different solvers is shown in Fig. 3. Even though the Adam solver achieves a lower loss in the early training stages, all algorithms show a steady progress and the SGD even overtakes the Adam solver at the end. Additionally we measure the average reward per episode obtained by the networks during training by evaluating the vehicle for 5 episodes every 10k iterations with $\epsilon = 0$. The abortion criteria are the same as during regular training. The results shown in Fig. 4 demonstrate that all optimization methods achieve a high average reward early on and become more stable with continued training. We conclude that the evaluated approaches are suitable for our DQN and the performance differences are not significant over a longer training period.

D. Reward function

Key to reinforcement learning is to define a proper reward function that allows the agent to rate decisions and determine the value of actions taken. In general, the reward is not necessarily a direct result of the last chosen action, but rather the result of the past several actions. The goal of our approach is letting the agent learn to find and follow the right-hand lane and drive around the track without leaving the lane.

As we can not rely on an existent intrinsic reward in our simulation environment, we define a suitable reward function to reach this goal. For the ground truth we use the lanelet representation [33] which allows us to get the distance to and from the target lane's boundaries and its center line from the car's current position. We construct two different reward functions: a naïve *distance-based* version and an extension adding *action-based* rewards.

1) Distance-based reward: A straightforward choice is to calculate the reward based on the distance d of the vehicle's geometric center to the center of the right-hand lane. We choose to apply the distance-based reward function r_{dist} with

$$r_{dist}(d) = 2 - (|d| + 1)^{\gamma}, \quad \gamma > 0$$

to reward or punish the agent accordingly. If the vehicle is positioned directly in the center of the lane, a maximum reward of $r_{dist} = 1$ is achieved, with a symmetric drop-off to both sides. The function's leniency towards higher distances can be adjusted with the parameter γ. For our results we use a rather aggressive punishment for leaving the lane, setting $\gamma = 6$.

While the distance-based approach is a good start, it does not fully capture what we would judge as "good driving". This is mainly due to the fact, that the orientation towards the lane is not taken into consideration. An agent wiggling around the center of the lane would generate the same reward as one that drives parallel to the center line with a fixed distance, while the driving experience would be contrasting.

2) Action-based extension: Due to the characteristic of the problem of no immediate feedback for an action, we examine several action-based additions to the distance-based reward. These action-based rewards can be motivated from a perspective regarding potential future use in a real world learning approach. We design the reward extensions to emulate a safety driver intervening in a vehicle's action to prevent unwanted or even dangerous behavior. Each extension defines a set of state-action pairs (x, a) for which adjustments are applied to.

First, we decrease the reward for state-action pairs, if the action a in state x results in an increase of the angle ϕ between the lanelet's direction and the car's heading. The set of those action-pairs is defined as $\mathcal{A}$. This adjustment results in better handling of curves, especially in curves with small radii.

To improve the driving experience on straight lane sections and reduce wiggling, we add an additional reward, if the agent chooses to drive straight on those sections while being within the distance $d_{\mathcal{S}}$ to the center line of the lane and the current ϕ is below the threshold $\phi_{\mathcal{S}}$ (defined as $\mathcal{S}$). In our experiments we set $d_{\mathcal{S}} = 0.16$ and $\phi_{\mathcal{S}} = 0.02$.

As a third action, counter-steering in curves is punished to give an incentive to reduce wiggling in curves as well (defined as $\mathcal{C}$).

Situations for these three reward adjustments are mutually exclusive and the action-based reward component is chosen according to the following formula:

$$r_{action}(x, a) = \begin{cases} +1, & (x, a) \in \{\mathcal{S}\} \\ -1, & (x, a) \in \{\mathcal{A}, \mathcal{C}\} \\ 0, & else \end{cases}$$

To properly balance the distance-based and the action-based components, we limit the distance-based reward at -2. The resulting overall action-based reward is then defined as

$$r = max(r_{dist}, -2) + r_{action}.$$

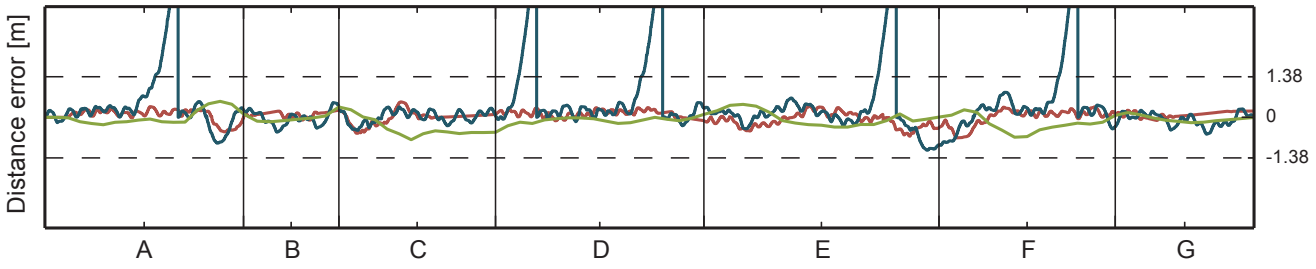
Those combined reward components are henceforth referenced as action-based reward.

IV. EVALUATION

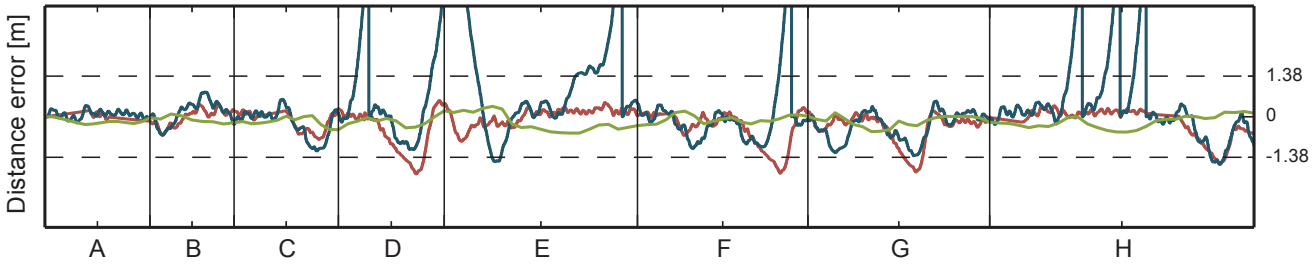
The evaluation of the vehicle agents trained with the reward functions described in Section III is performed on different tracks. We show that while the training with the naïve distance reward is insufficient to keep the vehicle on lane, an agent trained with the action-based reward function reaches an error value comparable to human performance.

A. Experimental Setup

For evaluation purposes, we designed two closed-loop tracks in our simulation environment (see Fig 2). The agents were trained on the left track, which has a slightly simpler layout. The right track was solely used for evaluating the agents. During evaluation the vehicle is starting at the beginning of the tracks' section A. The vehicle is reset to this starting position at the beginning of each run and then accelerates to a constant velocity. Starting position and



(a) Training track



(b) Evaluation track

Fig. 5: Signed distance error of the trained agents on the training track (a) and evaluation track (b). Labels on the x-axis mark the respective track sections (see Fig. 2). The lane boundaries are represented by the horizontal dashed lines. The agent solely trained with the distance-based reward (blue) leaves the lane and has to be reset multiple times on both tracks. The agent trained with the action-based extensions (red) and the human driver (green) achieve comparable results.

driving direction are indicated through the green triangle. An evaluation run is finished as soon as the agent completes an entire lap on the track. If the distance of the vehicle to its designated lane's boundaries exceeds the lane width, it is reset onto the closest point on the course.

For the evaluation we trained two DQNs using a SGD solver with the parameters described in Section III. The first network was trained utilizing the naïve distance reward function and the second with the action-based reward. Both networks were trained for 6.5 million iterations, which took about 36 hours with our simulation setup running in real time.

The driving performance of the trained vehicle agents is measured by their signed distance to the center line of the right lane. This distance error is calculated using the car's position given by the simulation environment and the lane boundaries provided via the generated lanelets. We use this measure – instead of using for example lap completion times – to discourage cutting curves and emphasize proper lane following.

To assess the obtained error values for the trained agents, we recorded human driving on the same tracks providing the same visual input as was used by the DQNs. Human driving was performed with a Logitech G25 Racing Wheel, which, in contrast to the discrete actions of the agents, sends continuous steering angles to the simulated vehicle model.

The system used to perform the training and evaluation is a computer running Kubuntu 14.04 equipped with an Intel i7 5820K CPU, a NVIDIA GTX 980 Ti and 16 GB RAM.

B. Results

The results of the aforementioned evaluation runs are shown in Fig. 5. As can be seen the agent trained with the naïve distance-based reward function is not capable to stay in lane for the complete track. In some curves it leaves the lane boundaries and has to be reset onto the course multiple times. Furthermore, the vehicle agent wiggles from left to right while driving, indicating the agent scanning for and reacting on lane boundaries. We assume that leaving the lane in curves might very well be due to this wiggling behavior, which results in the camera input losing sight of the lane markings.

The agent trained with the action-based reward on the other hand displays fewer and less pronounced wiggling motions. Especially on straight sections (c.f. Section G on the training track or Section A in the evaluation track) we observed the agent align with the lane heading and keep straight. Although it occasionally deviates from the center line in narrower curves, it manages to finish the courses without completely leaving the lane boundaries.

	Training Track	Evaluation Track
distance-based	0.552	0.769
action-based	**0.167**	0.335
human	0.211	**0.198**

TABLE II: Average distance error in m.

The graphs in Fig. 5 indicate that for most track sections the agent trained with the action-based reward performed on par with the human driver. The absolute mean error on

the training track even being marginally better than human performance and just slightly worse on the evaluation track as can be seen in Tab. II. The improvements observed compared to the purely distance-based reward function suggest our extensions to the reward enhance the agent's ability to learn how to drive.

During the evaluation the agents were able to process and select actions with a frequency of 200-250Hz. A video of the trained agents for further reference is provided online.[1]

V. CONCLUSIONS

We presented an approach for learning to steer a vehicle in a simulation environment using a Deep Q-Network (DQN). This environment is a realistic physics simulation working with Ackermann steering commands that are generating by the trained DQN based on visual input from a front facing camera. Given this input, the agent's goal was to keep on and follow the lane to complete laps on arbitrary courses. We designed an action-based reward to capture convenience factors motivated by human driving. Additionally, those action-based rewards were motivated by possible future integration into real world reinforcement learning scenarios using a safety driver's input analogue to the described actions given certain scenarios. Our results show that this reward greatly improved the learned driving behavior compared to a naïve distance based reward.

A current limitation of our approach is the discrete action space consisting of 5 steering angles values, prohibiting fine-grained adjustments of the vehicle's direction. This can be improved by simply increasing the number of actions or even using a continuous action space. Alternatively delta values instead of fixed steering angles can be used. Additionally the driving task can be made more challenging by adding intersections, static or dynamic obstacles to the tracks to further move towards a realistic learning environment.

Our network architecture is currently quite simple and will be extended, e.g. by using a double DQN or a dueling network. The training procedure can also be augmented by employing a prioritization of replay transitions or a more intelligent exploration instead of an ϵ-greedy approach, like Thompson sampling. To further increase the ability of our vehicle agent, one possibility is to use eligibility traces to track long term effects for chosen actions.

ACKNOWLEDGMENTS

The research leading to these results was conducted within the Tech Center a-drive. Responsibility for the information and views set out in this publication lies entirely with the authors.

REFERENCES

[1] D. A. Pomerleau, "Alvinn: an autonomous land vehicle in a neural network," DTIC Document, Tech. Rep., 1989.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016.

[3] Ö. Ş. Taş, F. Kuhnt, J. M. Zöllner, *et al.*, "Functional system architectures towards fully automated driving," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2016, pp. 304–309.

[4] C. Chen, A. Seff, A. Kornhauser, *et al.*, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," *Iccv2015*, 2015.

[5] Y.-W. Seo and R. R. Rajkumar, "Utilizing instantaneous driving direction for enhancing lane-marking detection," in *Intelligent Vehicles Symposium Proceedings*, IEEE, 2014, pp. 170–175.

[6] J. Huang, H. Liang, Z. Wang, *et al.*, "Robust lane marking detection under different road conditions," in *Robotics and Biomimetics (ROBIO)*, IEEE, 2013, pp. 1753–1758.

[7] L. C. León and R. H. Jr., "Car detection in sequences of images of urban environments using mixture of deformable part models," *Pattern Recognition Letters*, vol. 39, pp. 39–51, 2014.

[8] C. M. Bautista, C. A. Dy, M. I. Maỳalac, *et al.*, "Convolutional neural network for vehicle detection in low resolution traffic videos," in *Region 10 Symposium (TENSYMP)*, IEEE, 2016, pp. 277–281.

[9] S. Zhang, R. Benenson, M. Omran, *et al.*, "How far are we from solving pedestrian detection?" *arXiv preprint arXiv:1602.01237*, 2016.

[10] S. Paisitkriangkrai, C. Shen, and A. van den Hengel, "Pedestrian detection with spatially pooled features and structured ensemble learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 6, pp. 1243–1257, 2016.

[11] S. Houben, J. Stallkamp, J. Salmen, *et al.*, "Detection of traffic signs in real-world images: the german traffic sign detection benchmark," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, 2013, pp. 1–8.

[12] S. K. Berkaya, H. Gunduz, O. Ozsen, *et al.*, "On circular traffic sign detection and recognition," *Expert Systems with Applications*, vol. 48, pp. 67–75, 2016.

[13] N. Fairfield and C. Urmson, "Traffic light mapping and detection," in *Robotics and Automation (ICRA)*, IEEE, 2011, pp. 5421–5426.

[14] M. Weber, P. Wolf, and J. M. Zöllner, "Deeptlr: a single deep convolutional network for detection and classification of traffic lights," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2016, pp. 342–348.

[15] B. Huval, T. Wang, S. Tandon, *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[16] H. Yu, Y. Yuan, Y. Guo, *et al.*, "Vision-based lane marking detection and moving vehicle detection," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on*, IEEE, 2016, pp. 574–577.

[1] http://url.fzi.de/dqn-iv2017

[17] A. Asvadi, C. Premebida, P. Peixoto, *et al.*, "3d lidar-based static and moving obstacle detection in driving environments: an approach based on voxels and multi-region ground planes," *Robotics and Autonomous Systems*, vol. 83, pp. 299–311, 2016.

[18] P. Amaradi, N. Sriramoju, L. Dang, *et al.*, "Lane following and obstacle detection techniques in autonomous driving vehicles," in *Electro Information Technology (EIT)*, IEEE, 2016, pp. 674–679.

[19] F. Kunz, D. Nuss, J. Wiest, *et al.*, "Autonomous driving at ulm university: a modular, robust, and sensor-independent fusion approach," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2015, pp. 666–673.

[20] D. A. Pomerleau, "Neural network perception for mobile robot guidance," 1992.

[21] J. Koutnik, G. Cuccu, J. Schmidhuber, *et al.*, *Evolving Large-Scale Neural Networks for Vision-Based TORCS*, 2013.

[22] J. Koutník, J. Schmidhuber, and F. Gomez, "Evolving deep unsupervised convolutional networks for vision-based reinforcement learning," *Conference on Genetic and Evolutionary computation - GECCO '14*, pp. 541–548, 2014.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, pp. 1–14, 2015.

[26] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv*, pp. 1–28, 2016.

[27] T. Schaul, J. Quan, I. Antonoglou, *et al.*, "Prioritized Experience Replay," *arXiv*, 2015.

[28] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461 [cs]*, 2015.

[29] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv*, 2016.

[30] M. G. Bellemare, Y. Naddaf, J. Veness, *et al.*, "The arcade learning environment: an evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013.

[31] M. Zofka, F. Kuhnt, R. Kohlhaas, *et al.*, "Simulation framework for the development of autonomous small scale vehicles," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, IEEE, 2016, Forthcoming.

[32] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems*, IEEE, vol. 3, 2004, pp. 2149–2154.

[33] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: efficient map representation for autonomous driving," in *Intelligent Vehicles Symposium Proceedings*, 2014, pp. 420–425.

[34] Y. Jia, E. Shelhamer, J. Donahue, *et al.*, "Caffe: convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[35] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, 2009.

[36] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.