Deep Reinforcement Learning with Continuous Control in CARLA

Moritz Zanger[1] Florian Rottach[1] Izel Kilinc[1]

Abstract— In the course of this project, three policy gradient reinforcement learning algorithms are implemented and evaluated using OpenAI Gym's CarRacing-v0 environment and the open-source urban driving simulator CARLA. To reduce sample complexity, four variants of input representations with differing dimensionalities are employed, of which the bird's eye view representation of ground truth segmented images delivers the best results. We are able to show that a top-down input representation is a feasible concept for a vehicle in a real-world scenario. As a result, we propose a reward function that enables stable lane-keeping behavior and smooth driving in CARLA.

I. Introduction

The task of teaching vehicles how to drive autonomously in urban scenarios is a challenging and complex one to solve. Not only is there the problem of finding the adequate response to a given situation but also the challenge of taking into account the surrounding factors that have an influence on the state that a vehicle is in and its possible actions. To date, most approaches focus on the manual design of behavioral policies, such as defining a driving policy through the use of annotated maps [1]. While these solutions might work in situations which are documented by the provided mapping infrastructure, they are often difficult to generalize or scale, as they do not necessarily enable the comprehension of any given local scene. In order to make autonomous driving truly feasible in a real-world scenario it would be better to develop systems which are able to find their way without having to rely on an explicit set of rules. One possible solution to this task is provided by reinforcement learning methods. Here, the agent, i.e. the vehicle, actively searches for the optimal driving policy whilst trying to maximize a numerical reward signal. As opposed to imitation learning techniques, which have been popular in finding driving policies [2], reinforment learning algorithms enable a car to exceed human abilities, if applied correctly. In recent years, deep reinforcement learning methods have proven to be succesful in solving complex tasks such as playing GO [3] or Atari [4] and there have been efforts in tackling various problems in the field of autonomous driving, including continous control tasks [5].

However, two major drawbacks of reinforcement learning methods are their heavy depency on adequate input state representations [6] and, as with other machine learning techniques, their need of a sufficient amount of accurate sample data to train on. In order to be able to train safely on an adequate amount of data, one approach is the use of data from other domains. For this purpose, the urban driving simulator CARLA has been developed, which is used as a simulation environment for this project.

In previous projects, CARLA has been used to learn policies that allow agents to navigate through complex urban environments with imitation learning methods [7][8]. In this paper, it serves as an environment to solve a continuous control task via deep reinforcement learning. Several state-of-the-art reinforcement learning algorithms are implemented and compared, with regard to their performance considering driving tasks of increasing complexity. Additionally, reward functions for the respective problems are tested and different input represantations are designed and evaluated.

II. Related Work

Various reinforcement learning tasks have been tackled successfully in the past, leading to notable advances in learning policies for simulated and real-world robotic control problems [9][10] or solving complex game structures [11]. The resulting deep reinforcement learning methods have recently been developed so far as to achieving similar results or beating human experts in some of these tasks [3][12]. In their paper, Silver et al. introduce a new approach of teaching an agent the game of GO by combining Monte Carlo simulation with value and policy networks to evaluate board positions and choose new moves. In autonomous driving, Wolf et al. have achieved notable results using a Deep Q Network to evaluate possible steering actions from an action space of five predetermined steering options [13].

However, many tasks that could be solved with reinforcement learning approaches have continuous action spaces, to which approaches such as the DQN cannot be directly applied in a feasible manner [5]. Therefore, algorithms that can handle these high-dimensional action spaces have been developed in recent years. Mnih et al. present asynchronous variations of standard reinforcement learning algorithms and report the Asynchronous Advantage Actor-Critic (A3C) to have achieved the best results in their evaluations, including a physics control task with continuous action space in the MuJoCo Physics Simulator [14]. Lillicrap et al. introduce a different algorithm called Deep Deterministic Policy Gradient (DDPG), which they also evaluate in simulated physical environments, receiving near perfect results [4].

[1] The authors are with FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany {uvday, uydrn, uzdnz}@student.kit.edu

DDPG is also used by Kendall et al. to teach a full-sized autonomous vehicle a lane-following policy in an on-board manner. While already achieving good results with DDPG alone, they show that the additional use of a Variational Autoencoder greatly improves the overall performance [2], suggesting an improved state representation as an area of possible further development in reinforcement learning for continuous control tasks. Another proposition to reduce the complexity of visual features that has been investigated in several projects is the representation of the input images as a top-down view on the vehicle [15][8][16].

We extend this research by applying the aforementioned algorithms to autonomous driving tasks and evaluating their performance in OpenAI's CarRacing-v0 environment and in the CARLA driving simulator while investigating input states of differing complexity, such as bird's-eye view or latent space representations [17].

III. Background

We regard the typical reinforcement learning setting where an agent interacts with an environment $\mathcal{E}$. At each one of a number of discrete timesteps t the agent decides on taking an action a_t from a given set of actions $\mathcal{A}$. This is done based on the state s_t that the agent is currently in and following a policy π, which is a mapping of the possible states to the action space $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. In this case, π is stochastic, as it returns the probability distribution over the possible actions, and the action space is continuous.

As a result, the agent receives a reward r_t and the subsequent state s_{t+1} of his environment. The setup is assumed to follow the properties of a Markov decision process, where besides state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $r(s_t, a_t)$, we also include the transition function to the future states $p(s_{t+1}|s_t, a_t)$. The objective of the agent is to maximize his expected return R_t, which is the cumulative reward of the current state s_t and the rewards of future states, discounted with a factor $\gamma \in [0, 1]$.

In actor-critic methods the critic calculates the action value $Q^\pi(s, a) = E[R_t|s_t = s, a]$ describing the expected cumulative return for taking action a in state s under the given policy π, thus evaluating the selected action [18]. The actor on the other hand estimates an optimal policy, following $\pi(s) = argmax_a Q(s, a)$. Similar to the action value, $V^\pi(s) = E[R_t|s_t = s]$ defines the value of state s under the policy π and simply describes the expected return for pursuing a policy π from state s.

In our project, we consider a continuous control task in the context of autonomous driving where three different actions can be selected: steering in a range of [-1,1], throttle and acceleration in a range of [0,1] respectively.

IV. Concept/Methods and Models

In the course of this project, three differnt algorithms are used. Following the work of Lillicrap et al. [5] and Mnih et al. [14], the DDPG and A3C algorithms are implemented and compared according to their performance in the Gym environment CarRacing-v0. Later on, the PPO algorithm is applied to solve a continuous control task in the CARLA simulator.

The concept for our project consists of four major fields of interest. 1. Evaluating the performance of the implemented algorithms in order to find the reinforcement learning model which resolves our task best. 2. successfully defining a reward function that will teach the agent how to drive. Then, increasing the complexity of the reward function to driving in the right lane and, finally, driving in the lane without crashing. 3. Finding the optimal input state representation by varying the levels of realism of the input images. 4. Trying out the following three terminal conditions: Reaching more than 2000 steps, crashing and exceeding a certain distance to the center line of the road.

A. DDPG

One notable advance in reinforcement learning has been made by the development of the Deep Q Network (DQN) [12]. The DQN is able to solve tasks with high-dimensional observation spaces. However, it is only efficiently capable of working with discrete and low-dimensional action spaces. In order to adapt a DQN for the successful use with continuous control problems, as given in CarRacing-v0, a discretization of the action space has to be carried out, which can lead to two main difficulties: an explosion in the number of possible actions and the loss of important information [5].

To evade these obstacles Lillicrap et al. propose a novel approach, the Deep Deterministic Policy Gradient, which is a model-free, off-policy actor-critic algorithm. They adopt the advantages of DQN and combine them with the actor-critic framework, resulting in the stabilization of Q-learning through the usage of a replay buffer and soft updates on the target networks θ' of both actor and critic, through

$$\tau << 1 : \theta' \leftarrow \tau\theta + (1 - \tau)\theta', \tag{1}$$

as well as finding a deterministic policy.

In order to enable better exploration, we add noise in the form of the Ornstein-Uhlenbeck process to our policy. For our continuous control problem, we chose the Ornstein-Uhlenbeck process due to the fact that it produces temporally correlated noise, which enables smoother transitions, compared to e.g. Gaussian noise.

B. A3C

The Asynchronous Advantage Actor-Critic (A3C), introduced by Mnih et al. [14], has become a go-to algorithm in Deep Reinforcement Learning due to its performance, robustness, and ability to perform well on high-dimensional action and state spaces. A key characteristic of A3C is the utilization of multiple agents, each equipped with its own environment instance and its own set of network parameters. One of the advantages of this approach is the diversification of the collected experience but yields the challenge of handling gradient update mismatches between the asynchronously collected network parameter updates. In our implementation of A3C we adapt several alterations as opposed to the original version by Mnih et al.. Despite being an on-policy method, we include a numerically efficient n-step return as proposed by jaromiru [19].

Furthermore, experiences are buffered in a global update queue and updates are only performed by a master network. This feature helps decorrelate experiences, emphasizes exploration and allows for more GPU-friendly batch-learning. However, one has to keep in mind that this contradicts the original update rule by Mnih et al. and might lead to policy lag. A proper update frequency in the master network is therefore of major importance.

C. PPO

The Proximal Policy Optimization (PPO) algorithm was developed to improve training stability by avoiding parameter updates that excessively change a given policy at each step. The algorithm that PPO is based on is called Trust Region Policy Optimization (TRPO) and realizes training stability through a KL divergence constraint on the policy update range at each iteration [20]. PPO simplifies this concept by incorporating the hard constraint into the objective function through Lagrangian duality and, thus, softening the constraint. Additionally, PPO clips the objective function if the ratio between old and new policy lies outside the range of [1-ϵ, 1+ϵ], which discourages large policy changes. Tests on benchmark tasks have shown PPO to achieve outstanding results while reducing TRPOs complexity significantly [21].

D. Reward function

The design of the reward function is of major importance to the outcome of this project while being a much more complex process than initially expected. In Gym's CarRacing environment good results could be achieved in the end by employing the rather simple and already provided reward function to our model. For CARLA however, it was necessary to invest more time into the engineering and design of a suitable incentive system, as the driving behavior in this environment depends on larger variety of factors and the performance is not only measured by the speed of the agent. IThe process of finding a suitable reward function comprises the following tasks:
In the first step, we investigate possible sensor and measurement inputs that might have an impact on the driving behavior and discuss their impact. As summarized in Table 1, the outcome consists of eight possible features, that are subsequently tested.

TABLE I: Identified reward function components

Component	Description	Intention
Per frame penalty	The penalty amount subtracted from the reward for each frame	Forces agent to move in order to compensate the penalties
Lane invasion increment	Is either 0 or 1 and reflects if a lane invasion took place for the current frame.	The agent should perform as few lane changes as possible
Steering angle	The absolute angle of the steering wheel	Avoids oszillations in the driving behavior
Delta heading	The angle between current street direction and car position	Agent should drive as straight as possible relatively to road direction
Position change	The angle between current street direction and car position	Maximize travelled distance
Collision binary per frame	Penalty for crashing into other vehicles, objects or road infrastructure	Avoid crashes and improve security of driving
Velocity	The agent's absolute speed retrieved from the CARLA engine	Forces the agent to move forward
Distance to middle lane	The absolute distance to center in meters - can also be squared to improve	Drive as centered as possible

We start with incrementally adding these attributes to our reward calculation leading to the realization that the main challenge is adjusting the weights and harmonizing the contrary effects of the terms. For example, giving the velocity a relatively high weight, such as 0.8, while giving the distance to center line a weight of 0.2 results in an agent that speeds over the map and pays only little attention to lane invasions. On the other hand, the agent will drive very slowly or not at all, if the rewards for oscillations are too high compared to the velocity. Considering, that we have not only two, but several possible components, this results in a complex combinatorical problem that can either be solved by trial and error or by applying permutational optimization techniques. Initial results are achieved by finding a reward function through the former method.
The above list is reduced by applying the following considerations that result from testing different approaches. Firstly, some components show a redundant behavior and, hence, one of them can be removed. An example would be the velocity and the position change - both contain the same information. Furthermore, the lane

invasion and steering angle attributes do not affect the driving behavior in a positive way. The two most important parameters turn out be the velocity and the delta heading, which expresses the relative angle to the current street angle. Our final reward function and the aggregation weights can be found in table 2.
We want to note here, that a performance-wise comparison between the different parameter combination is hard to measure with a metric and is not the aim of this project. However, it would make sense to develop a suitable approach to categorize and assess the performance of different reward functions towards the desired outcome.

TABLE II: Final reward function terms and weights

Component	Effect	Weight
Per frame penalty	Lead to a strong initial learning behavior. Already after few steps the agent kept accelerating to compensate this penalty.	-0.01
Velocity	When selected too small, slow learning and when too high, strong lane oscillations / non-smooth driving.	+0.05
Delta heading	Introducing this term improved the driving stability enormously.	-0.005
Squared distance to middle lane	Squaring had very beneficial effects, other powers were to restrictive.	-0.01
Collision binary	Might be set to an even higher value but lead to attempts to avoid other objects	-100

The result, in combination with tuning the reinforcement learning models, is very promising and can be summarized as follows: The agent is capable of driving smoothly within the right lane and can perform turn maneuvers on most intersections. It attempts to drive around other vehicles, but is not capable of braking. Different models were trainde on this reward function and all of them had a good performance compared to other incentive attempts. We assume, that even better results can be achieved by applying an improved optimization method to find the best parameter combinations. However, this is a combinatorical problem and requires several simulation runs.

E. Input representation

Dealing with the high-dimensional environment in CARLA is one of the central challenges in implementing a well functioning RL algorithm. In consideration of an abundance of possible sensor types available in CARLA we pursue increasing levels of realism which generally also correspond to increasing levels of difficulty. These aforementioned levels of realism involve the following:

- Ground truth segmented bird's eye view
- Ground truth segmented front view
- Latent space generated from ground truth segmented images
- Latent space generated from rgb images

Notably, these input representations differ in dimensions and thus lead to different network architectures in the appended network architectures of the RL agent.

a) Ground truth segmented bird's eye view: The Ground truth segmented bird's eye view is a rather unrealistic scenario, in which we assume availability of a camera positioned 20m above the performing agent. It is merely imaginable in a fully observed city where autonomous driving is part of a high-level traffic control system. Though rather unrealistic, we implement a 13-class ground truth segmented bird's eye view due to its similarity to the CarRacing environment and its obvious advantages in containing information central to navigational tasks. The full list of the included classes are described by Dosovitskiy et al.[17]. We deem a 1-channel grayscale image with size 64x64 large enough to contain key information for the algorithm (fig. x l.)

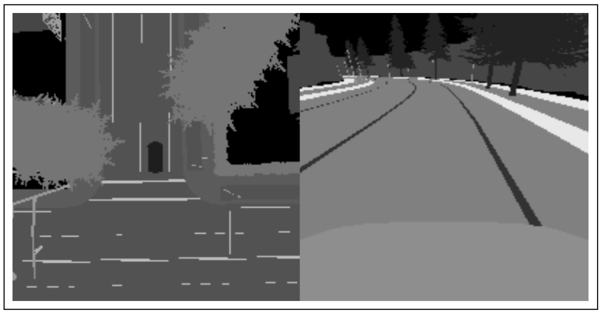

Fig. 1: left: Ground truth segmented bird's eye view image, 64x64 in grayscale, 9 classes right: Ground truth segmented front view image, 80x80 in grayscale, 9 classes

b) Ground truth segmented frontview: Much like the ground truth segmented bird's-eye view, the corresponding front view input representation assumes the availability of a perfect segmentation camera. Nonetheless, we increase realism with this representation as the camera is placed in front of the car. Again, we choose a 13-class, 1-channel grayscale image with a slightly increased size of 80x80 to account for a larger view angle of the camera.

c) Latent space generated from ground truth segmented images: In this section we further discuss a modified version of the integrated encoder-decoder network that is based on Dziubinski's [22] implementation. The underlying idea of this model is to guide feature extraction towards more useful, problem-specific features by exposing the model to the additional target of reconstructing a bird's-eye view from the vehicle-based camera images. In its original architecture, the network comprises five input tensors and seven output tensors with the inputs representing images of a front-, rear-, left-, right-, and top-view camera. The model is built with the Keras functional API and generally serves two purposes. On the one hand, each input tensor is encoded into a 64-sized feature vector and decoded into its original shape with a cross entropy loss with regard to the original input. This is achieved with separate branches of autoencoders with three convolutions respectively. On the other hand, a separate generative branch creates a bird's-eye view reconstruction based on the concatenated feature vectors of the vehicle-based cameras. In addition to the cross entropy loss at the reconstructed bird's-eye view output, a mean squared error loss is calculated against the output of a subtract layer between the autoencoder's feature vector und the reconstructed feature vector to support convergence. Notably, the autoencoder branch of the top-down camera view is solely purposed for improving the latent space of the generative branch during training and is thus not required for inference.

In comparison to Dziubinski's [22] vanilla version, we adjust the architecture to fit the single camera bottlenecks into a length 64 vector and the reconstructed bird's-eye view bottleneck into a length 128 vector. To reduce complexity, we condense both input and target segmentation images to three classes and implement a generalized weighted cross entropy[23] loss function to account for the unbalanced distribution of vehicles and obstacles.

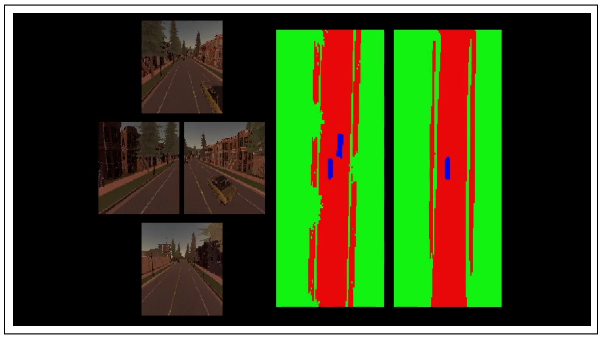

Fig. 2: left: rgb images from 4 vehicle-based cameras
2nd from right: ground truth segmented bird's eye view
right: reconstructed bird's eye view

d) Latent space generated from rgb images: To deal with the higher input complexity in RGB-images, we extended the encoder and decoder models to 5 convolutions. Training data was collected both manually and by autopilot to retrieve a better balance in driving styles, speed and locations. During training, the maximum of 308 Vehicles were spawned in the environment to increase vehicle-class occurrence in the data.

F. Training

In order to align the algorithms with the varying input dimensions we had to implement a wrapper around the actual models that adjusts the network architecture accordingly. Especially the fact that the latent space is flattened and not 2-dimensional such as the ground truth pictures made it neccessary to implement convolutional as well as fully connected networks. Luckily, for the stable-baselines PPO model this is implemented very quickly. The architecture of our Keras-DDPG model however, had to be adjusted completely. We performed the training in four process steps, aligned with the input representation types mentioned previously. Therefore, the initial training was carried out on the ground-truth front view, afterwards on the ground-truth bird's eye view and finally we trained on the latent space of rgb and ground-truth segmented images.

1) DDPG: The Deep Deterministic Policy Gradient performed well on our initial attempts within gym CarRacing and we identified reasonable hyperparameters for proper learning. We applied the same model on the new input features of CARLA and came to the sobering result that the input seemed to be too complex for the model. We assume, that the learning algorithm was distracted by the advanced simulation environment. The following illustration emphasizes the poor behavior of the model (trained on circumstances where other models performed well). Hence, we decided to continue with different learning approaches and agreed due to recent developments on the PPO.

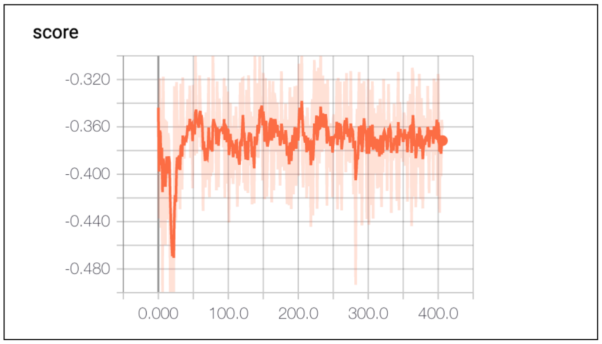

Fig. 3: Keras DDPG learning behavior

2) PPO: The stable-baselines PPO2 model is easy to train and only few hyperparameters need to be adjusted. The drawback of this approach is that customization of these pre-defined models is rather laborious. Especially a combination of diverse input types such as scalars and images requires arduos architecture adjustments. We changed the following parameters and arrived at a very fast and crisp learning behavior that enables the agent to start driving after already very few episodes.

The learning behavior usually increased until 200k steps and started to stagnate afterwards. The achieved episode rewards and entropy loss over several runs was very stable and performance drops occured rarely. The

TABLE III: Final reward function terms and weights

Parameter	Default Value	Selected Value
Learning rate	0.00025	0.0004
Clip range	0.2	0.1
Gamma	0.99	0.97
N_steps	128	1024
Environment steps	25k	200k

results on the proved input representations were very different: We trained the PPO with the same reward function, the same amount of steps at the same environment spawn point and hence assumed equal conditions. The front-cam-view resulted in a very oscilating driving behavior and the rewards were damped by the penalties for inaccurate delta heading. The birds-eye-view on the other hand had a very smooth driving behavior and deliverd the best performance in our test. This comes from the improved view on the street trajectory from above. Due to training capacity, we decided to restrict training to the feature vector of the RGB encoded model. Considering the easier and smaller latent space representation we realized that the algorithm trains very fast but can't reproduce the performance of the ground-truth.

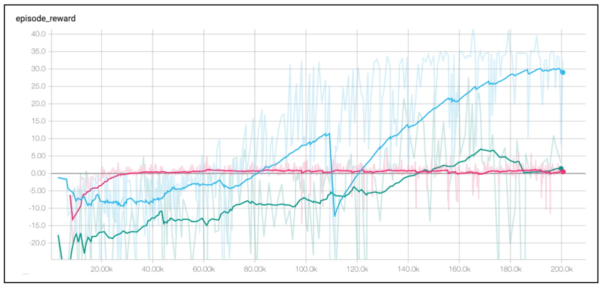

Fig. 4: Green line: Performance on front-cam-view, Blue line: Performance on birds-eye-view, Red line: Performance on latent space of RGB model

V. Results

A. CarRacing-v0

Due to the simple input structure and the straight-forward reward system we achieved stable results, even with using not the most-advanced learning algorithms (We consider the PPO as an improvement over the DDPG). Also the A3C could generate promising results (average rewards around 700) with only few training time. In total, these results can be explained by the simplicity of the environment. However, here we invested more time on the finetuning of the hyperparameters compared with CARLA.

B. CARLA

The CARLA simulation environment is far more complex than the previous environment we worked with. Since the application of the stable-baselines algorithms is rather straight-forward, the main challenge comprised the design of proper input representations and the optimization of terminal conditions, rewards and parameters.

TABLE IV: Project Goal: Input State Representations

Difficulty Level	Goal	Accomplished
Level 1	Ground truth segmented bird's eye view	✓
Level 2	Ground truth segmented front view	✓
Level 3	Latent space generated from ground truth segmented images	discontinued
Level 4	Latent space generated from rgb images	–

As can be seen in TABLE IV, we were not able to produce a high-performance agent using the latent space of the RGB-model, despite its promising training speed. We assume that the architecture didn't contain the complete information in the layers that were extracted in our experiment and that this was the reason, a similar outcome as the ground-truth birds-eye-view was not possible. One of the issues contributing to this, is the mismatch between the unconventional driving style of a learning agent and the driving style of a human/autopilot, at which data for the encoder-decoder model was collected. Despite using a weighted categorical cross entropy loss, the network struggled with reconstructing pixels of the vehicle and obstacles class. Aside from improving training data and network parameters, we consider the usage of more sophisticated encoder-decoder models such as Deeplab-V3 for feature extraction as promising. Moreover, we believe feature extraction models should rather be fine-tuned during the agent's training, than be used for inference only. This way, a trade-off between problem-specific optimal feature extraction and data/time efficiency can be pursued. Given the variety of available sensors in CARLA, additional input information such as 2D-Lidar data might further improve the agents ability to avoid crashes.

TABLE V: Project Goal: Driving

Difficulty Level	Goal	Accomplished
Stage 1	Driving	✓
Stage 2	Lane Following	✓
Stage 3	Driving with Traffic and Obstacles	×

TABLE V lists our overall project goals with respect to the driving behavior of our best-performing agent. The defined reward functions managed to teach our agent how to follow his lane with input states in ground truth bird's eye view representation. The agent actively tried to avoid crashes in some cases, however, we believe that further adjustments to the reward function could improve the results in this area. Terminating after 2000 steps or after a crash improved the agent's behavior while terminating when the agent moved too far from the center line had a negative impact on training results.

VI. Conclusions

As result of this practical seminar we have successfully applied different deep reinforcement approaches to the CARLA environment and finally achieved a stable and promising lane-keeping behavior for the agent. This was achieved by a combination of finding the most suitable input representation for smooth driving, fine-tuning of the rewardfunction and identification of the best performing hyperparameters. The outcome proves that reinforcement learning can be applied to complex environments as well as long as the desired behavior is incooperated in the reward system. Furthermore, we could prove that the birds-eye-view leads to a better performance as a single front camera and is also feasible in a real-world setup by generating the representation from four cameras surrounding the car. In the training process we recommend to not use early-termination for the episodes, especially if the reward function contains negative components, since this increases the probability that the agent starts to drive off the track as soon as possible. Finally, we emphasize that the driving behavior has the potential to be even further improved in the future by selecting better data, learning algorithms and an optimal reward function. We can also imagine, that the involvement of further attributes in the model such as the current velocity can be used to describe even more complex driving situations.

For the future we additionally suggest a combination of the birds-eye-view with a front camera in order to enable the identification of traffic signs and traffic lights, which wouldn't be possible from above. Also, we assume that the improvement of the encoder-decoder architecture could boost the learning speed and overall performance.

- Similar to the abstract but more detail
- Conclusion of the key points of each section
- Summary of main findings
- Important conclusions that can be drawn
- Discuss benefits and shortcomings of our approach
- Suggest future areas of research

References

[1] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 4, pp. 1135–1145, April 2016.

[2] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in Proceedings of the International Conference on Robotics and Automation (ICRA), 2019.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602.pdf

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971 [cs, stat], Sept. 2015.

[6] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," CoRR, vol. abs/1904.09503, 2019. [Online]. Available: http://arxiv.org/abs/1904.09503

[7] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, and V. Koltun, "End-to-end driving via conditional imitation learning," CoRR, vol. abs/1710.02410, 2017. [Online]. Available: http://arxiv.org/abs/1710.02410

[8] J. Chen, B. Yuan, and M. Tomizuka, "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety," CoRR, vol. abs/1903.00640, 2019. [Online]. Available: http://arxiv.org/abs/1903.00640

[9] S. Levine and V. Koltun, "Guided policy search," in Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ser. ICML'13. JMLR.org, 2013, pp. III–1–III–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042817.3042937

[10] T. de Bruin, J. Kober, and K. Tuyls, "The importance of experience replay database composition in deep reinforcement learning," 2015.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," CoRR, vol. abs/1511.05952, 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[13] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Härtl, F. Dürr, and J. M. Zöllner, "Learning how to drive in a real world simulation with deep q-networks," in 2017 IEEE Intelligent Vehicles Symposium (IV), June 2017, pp. 244–250.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," arXiv:1602.01783 [cs], Feb. 2016.

[15] M. Bansal, A. Krizhevsky, and A. S. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," CoRR, vol. abs/1812.03079, 2018. [Online]. Available: http://arxiv.org/abs/1812.03079

[16] N. Djuric, V. Radosavljevic, H. Cui, T. T. V. Nguyễn, F.-C. Chou, T. H. Lin, and J. G. Schneider, "Short-term motion prediction of traffic actors for autonomous driving using deep convolutional networks," 2018.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," arXiv preprint arXiv:1711.03938, 2017.

[18] "Policy gradient algorithms," https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#ppo, accessed: 2019-10-20.

[19] J. Janisch, "Let's make an A3C: Implementation," https://jaromiru.com/2017/03/26/lets-make-an-a3c-implementation/.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in Proceedings of the 32nd International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," CoRR, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[22] M. Dziubiński, "From semantic segmentation to semantic bird's-eye view in the CARLA simulator," https://medium.com/asap-report/from-semantic-segmentation-to-semantic-birds-eye-view-in-the-carla-simulator-1e636741af3f, May 2019.

[23] Z. Zhang and M. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," in Advances in Neural Information Processing Systems 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 8778–8788.