Deep Reinforcement Learning with Continuous Control in Driving Simulation

Patrick Hemmer[1], Tina Menke[1] and Mingyuan Zhou[1]

***Abstract*— We present a deep reinforcement learning approach with continuous control where an agent learns to drive a car in two different virtual environments using Proximal Policy Optimization (PPO). In the environment CarRacing-v0, the agent is able to learn a policy for following the race track by receiving raw RGB images as inputs which are fed to the algorithm through a shallow convolutional neural network. Using these results as a basis, we train an agent in the environment CARLA, that can drive on straight as well as curvy roads while staying within the designated lane markings. We achieve this by defining a reward function that positively rewards the distance traveled, whereas leaving the lane is penalized. The agent can master the desired tasks by learning a driving policy from cropped segmentation images fed through a deeper convolutional neural network. By conducting hyperparameter optimization, we stabilize the agent's training process while improving its ability to generalize simultaneously.**

I. Introduction

One of the most ambitious goals pursued nowadays is to make the dream of cars driving autonomously on a grand scale become reality. This dream is almost as old as the car itself. Not long after the invention of the first motor-powered car, people started to strive for a technical solution for autonomously driving cars. Lately, many approaches towards autonomous driving focus on addressing this task with supervised learning. This requires a large amount of training data to train a model that can generalize well. However, as driving involves intensive interaction with the environment addressing all possible scenarios with a supervised learning approach is almost impossible [1].

These challenges lead naturally to examining learning approaches that primarily focus on an understanding of the local scene. For this reason, in this paper, we apply deep reinforcement learning with continuous control to address the autonomous driving task.

Reinforcement learning is a paradigm within the field of machine learning that evolved through the psychological and neuroscientific perspectives on animal behavior. It is defined as goal-directed learning from interaction which involves learning what to do and how to map situations to actions in order to maximize a numerical reward signal. Over the past few years, this paradigm has gradually become one of the most active research areas in machine learning showing promising results in solving complex tasks [2].

In this context, [3] developed a deep reinforcement learning model to learn complex control policies for several Atari 2600 computer games outperforming previous reinforcement learning algorithms. On three of them, they were even able to surpass expert human player performance.

Despite this success, its application to the domain of autonomous driving has just started in the last years [4]. As humans learn to drive a car by observing the environment and adapting their behavior with the aim of an accident-free ride, the generality of reinforcement learning entails great potential in contributing significant advantages towards reaching the goal of autonomous driving.

As steering and accelerating can be modeled more precisely in a continuous instead of a discrete action space, we train a Proximal Policy Optimization (PPO) agent in two virtual simulation environments - CarRacing-v0 and CARLA and evaluate them subsequently in these environments. Conducting these experiments in the real world using a physical car is time-consuming and accompanied by high costs. Virtual simulation environments offer not only the possibility to train agents for many episodes time and cost-efficiently but also to prevent humans and vehicles from getting harmed or damaged.

II. Related Work

Reinforcement learning has achieved remarkable results in areas such as simulated robotics or at playing Atari computer games [5], [3]. Due to the limitations accompanied by addressing the autonomous driving task with supervised learning it has been applied increasingly to this area over the last years. As reinforcement learning agents learn through exploration by trial-and-error, training only in the real world would result in undesirable driving actions leading to possible damage to the vehicle and its environment. Therefore, most reinforcement learning research for autonomous driving is conducted in virtual environments.

[6] use a modification of an A3C model, Asynchronous Advantage Actor-Critic algorithm with continuous certainty. They train an agent to drive a car in the OpenAI Gym CarRacing-v0 environment which is a 2D car racing game with top-down bird perspective on the racetrack. They use RGB images as input which are processed to grey scale followed by zero-centering, cropping and stacking 5 consecutive pixel frames. The preprocessed images are fed as input into a CNN-network architecture. The features extracted by the network are flattened and put into the policy and value network. Their modified agent ranks fourth on the OpenAI Gym leaderboard.

[1]Equal contribution. The authors are with FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany.

[7] train a Deep Q-Network end-to-end to control a car in a 3D physics simulation environment. The model learns to steer the vehicle through camera images as input. Hereby, no labeled training data is needed. Evaluating the model, the agent achieves human-level driving performance on previously unseen routes in the simulation. However, one of the downsides of the approach is the usage of a discrete action space of the steering angles.

[8] address this limitation in their model by training a PPO agent with a continuous action space. In their approach, they train an agent to drive a car in the 3D open-source simulator CARLA. Its task is to learn a policy that controls only the steering whereas the throttle is controlled by a Proportional-Integral-Derivative (PID) controller set to constant speed. The policy receives as input a segmentation image. The agent is trained on several fixed routes with each route representing a different training scenario. However, they provide only a qualitative evaluation of the model's performance.

Whereas the models described in the previous papers are trained on simpler driving scenarios, [9] use a model-free deep reinforcement learning approach designed to address challenging urban driving scenarios. Using variational auto-encoding their model is capable to capture low-dimensional latent states. They apply Double Deep Q-Learning (DDQN), Twin Delayed DDPG (TD3) and Soft Actor-Critic (SAC) reinforcement learning algorithms to learn a policy to output correct control commands using the driving simulator CARLA. Evaluating the model, the agent is able to master the desired tasks.

Even though using reinforcement learning for autonomous driving in a virtual world has shown promising results, the goal should be to transfer these methods to the real world. [10] suggest a translation network approach to enable a model's functionality in the real world. The framework translates virtual images in the first step to their segmentations which are then translated to their real-world counterparts in the second step. Having trained an Asynchronous Advantage Actor-Critic agent with this approach results when tested on real-world data in an improved driving performance compared to the agent only trained on virtual images.

In contrast to the aforementioned papers, we pursue a twofold approach. In the first step, we aim at proposing a solution for the autonomous driving task in the CarRacing-v0 environment by examining different image-based inputs, network structures, and hyperparameters. Taking these results as a basis, we apply them in CARLA and train an agent that can stay within a designated lane on the road while taking turns along the way. Furthermore, we experiment with more complex driving situations including the presence of obstacles and other driving cars.

III. Concept

For the simulation, we use both the simulation environment OpenAI Gym CarRacing-v0 and CARLA. OpenAI Gym is a tool for developing and benchmarking reinforcement learning algorithms providing different environments such as CarRacing-v0 [11]. CARLA (Car Learning to Act) is an open-source 3D simulator for the development, training, and evaluation of autonomous driving systems [12]. As the key focus of this paper is autonomous driving through deep reinforcement learning with continuous control, besides the reinforcement learning algorithm, we have to define the desired driving task, the state space, the action space as well as the reward function for the CarRacing-v0 and CARLA model respectively.

A. Reinforcement Learning Algorithm – Proximal Policy Optimization

Concerning the reinforcement learning algorithm, we use Proximal Policy Optimization (PPO). It is a model-free reinforcement learning approach and belongs to the family of on-policy optimization methods using multiple epochs of stochastic gradient ascent for conducting each policy update [9], [13]. Model-free reinforcement learning algorithms are very general and can, in theory, learn any task [14].
One of its advantages is the simplicity of its implementation while being as stable and reliable as trusted-region methods. Moreover, the PPO algorithm has shown to be both sample efficient and relatively easy to tune.
The main objective of the PPO algorithm is a clipped surrogate objective:

$$L^{CLIP}(\Theta) = \widehat{E}_t \left[min(r_t(\Theta)\widehat{A}_t, clip(r_t(\Theta), 1-\varepsilon, 1+\varepsilon)\widehat{A}_t) \right] \quad (1)$$

The first term inside the min function is $\widehat{E}_t \left[r_t(\Theta)\widehat{A}_t \right]$.
$r_t(\Theta)$ denotes the probability ratio:

$$r_t(\Theta) = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}(a_t|s_t)} \quad (2)$$

Thus, $r(\Theta_{old}) = 1$. $\widehat{A}_t$ is the difference of the discounted rewards minus the baseline estimate:

$$\widehat{A}_t = Discounted\, rewards - Baseline\, estimate \quad (3)$$

In simpler words it can be described as "we know what happened" - "What did we expect would happen?".
The second term $clip(r_t(\Theta), 1-\varepsilon, 1+\varepsilon)\widehat{A}_t$ modifies the surrogate objective through clipping the probability ratio. This decreases the probability for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. Within the clip-function ϵ is a hyperparameter.
The minimum is taken from the clipped and unclipped objective. Hereby, the change in the probability ratio is ignored when it would enhance the objective and it is included when it would worsen the objective.
The surrogate losses can be computed by making use of a typical policy gradient implementation. As we are using a

neural network that shares parameters between the policy and value function, a loss function combining the policy surrogate and the value function error term has to be taken into account.
This results in the following objective that is maximized each iteration:

$$L_t^{CLIP+VF+S}(\Theta) = \widehat{E}_t \left[L_t^{CLIP}(\Theta) - c_1 L_t^{VF}(\Theta) + c_2 S[\pi_\Theta](s_t) \right] \quad (4)$$

L_t^{VF} denotes the squared error loss $(V_\Theta(s_t) - V_t^{targ})^2$, c_1 and c_2 are coefficients. S is an entropy bonus to ensure sufficient exploration [13].

B. Framework for CarRacing

CarRacing-v0 is a two-dimensional racetrack environment surrounded by grass and a car representing the agent.

1) Driving Task: In order to "solve" the CarRacing environment and win the game respectively the agent has to achieve an average reward of 900 over 100 consecutive episodes. One episode finishes once all tiles are visited. Thereafter, the agent is randomly spawned on the track for the next episode [11]. The driving task of the agent is to drive through the race track as fast as possible.

2) State Space: The state space consists of a 96x96 RGB image displaying the racetrack as well as the car from a top-down perspective. At the bottom of the image, there is a black bar showing the car's speed, four ABS sensors, the steering wheel position, and a gyroscope. Figure 6 in the Appendix displays an exemplary input image.

3) Action Space: As for the action space, the agent has to choose three continuous values. The steering wheel position ranges continuously from -1 (left) to 1 (right). Both throttle and the brake can be set to any value between 0 and 1 [15].

4) Reward Function: The reward function is defined as follows. The agent receives a negative reward of -0.1 every frame. For every track tile visited it is rewarded by +1000/N where N is the total number of tiles in the track. Every track contains approximately 300 track tiles. The agent does not receive a negative reward for going off the track other than the frame costs [16].

C. Framework for CARLA

As CARLA is a 3D simulation environment specifically designed for the development, training, and validation of autonomous driving systems, it provides different urban layouts, vehicle types and other traffic participants such as pedestrians and autonomous driving cars.

1) Driving Task: The driving task we want to accomplish consists of three subtasks with increasing difficulty. All subtasks are carried out in Town 5 which is provided by CARLA. The first task of the agent is to drive on a straight road within the lane markings. Secondly, the agent shall master to take turns while staying within the designated lanes. Thirdly, we construct more complex driving situations by adding autonomous driving cars and static objects to the world.

2) State Space: The information taken into account for the state space in CARLA is manifold. In contrast to CarRacing, the map in which the agent is located can be perceived through various channels. Technically, this is realized by equipping the car with various sensors. There are cameras such as RGB, depth, LIDAR and semantic segmentation and sensors like collision, lane invasion, obstacles, and global navigation satellite system (GNSS) position. Apart from the car's position, it is possible to retrieve information about its driving states such as velocity, acceleration, or distance to the centerline.

We use the RGB and semantic segmentation camera to retrieve input images that are fed into the convolutional neural network.
Moreover, we make use of the collision and lane invasion sensor as well as information about the car's position, velocity, acceleration, and distance to centerline. This information is used for the reward function (see III-C.4).

3) Action Space: The action space consists of the actions steering and operating the throttle. The steering wheel position ranges continuously from -1 (left) to 1 (right). Furthermore, the throttle can have a value in the interval [0, 1].

4) Reward Function: The general idea of the reward function's structure is derived from [12]. We design the reward function as a weighted sum consisting of the following five terms: traveled distance $\|p_t - p_{t-1}\|_2$, velocity difference $(v_t - v_{t-1})$, collision binary difference $(c_t - c_{t-1})$, intersection with lanes $(s_t - s_{t-1})$ and distance to centerline (d_t).

$$r_t = 5 \|p_t - p_{t-1}\|_2 + 0.05(v_t - v_{t-1}) - 2(c_t - c_{t-1}) - 5(s_t - s_{t-1}) - 2(d_t) \quad (5)$$

On the one hand, to encourage driving as well as accelerating, we weight the traveled distance and velocity difference with positive coefficients. On the other hand, the factors for collision, intersection with lanes and distance to centerline are weighted negatively to penalize the agent whenever these situations occur.

IV. Experiments

A. Training in CarRacing

We trained the PPO agent in the CarRacing-v0 environment using a high-quality implementation of the algorithm provided by OpenAI Baselines [17]. To boost the agent's performance, we preprocessed the input images and varied the degree of exploration.
Moreover, we modified the network structure of the underlying CNN which is used to extract features from the input images. All modifications just mentioned are described in detail below.

1) Network Structures : We used three different convolutional neural network (CNN) architectures through which the images are fed into the reinforcement learning algorithm. For the first one – CNN – we took the network architecture used by [18]. The second network architecture – CNN paper – was taken from [6], whereas the third one – CNN small – implements the architecture that can be found in the baselines folder of the gym library [17]. Figure 7 in the Appendix illustrates the three different CNN architectures visually.

2) Preprocessing: We preprocessed the aforementioned 96x96 RGB images to disregard unnecessary information. First, we applied grey scaling and zero-centering. Then, we cropped 6 pixels from each side out to remove the black bar at the bottom of the image. Lastly, we stacked 5 consecutive pixel frames to add information about the temporal change to the input. The preprocessing results in 84x84x5 zero-centered input images. Figure 1 depicts an exemplary image generated by applying the preprocessing method just described.

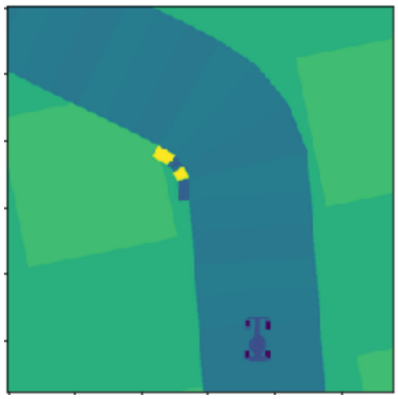

Fig. 1: Preprocessed input image.

3) Hyperparameter Tuning: The final objective function of the Proximal Policy Optimization algorithm (see equation 4) contains an entropy term which is weighted with the entropy coefficient c_2. The entropy coefficient determines the agent's degree of exploration. By increasing the value of c_2 the agent becomes more likely to choose a random policy and thereby explores its surrounding. We changed the value of c_2 from 0 to 0.01 aiming to find the right balance between exploration and exploitation.

With regard to the combinations resulting from the above-described modifications, we trained these model configurations four times over 1e6 timesteps. For all hyperparameters except the entropy coefficient c_2 we used OpenAI Baselines' default values [17].
Figure 2 displays the development of the average reward and its standard deviation during training using the default CNN, CNN small and CNN paper with raw 96x96 RGB images as input. The default CNN has the lowest average reward and a very high standard deviation. From this, we can conclude that the network architecture is too deep causing unnecessarily extensive feature extraction. The shallower network structures of the CNN small and CNN paper result in an average reward of approximately 500 after 1e6 timesteps. The CNN small has a smaller standard deviation than the CNN paper; thus, it performs best overall.

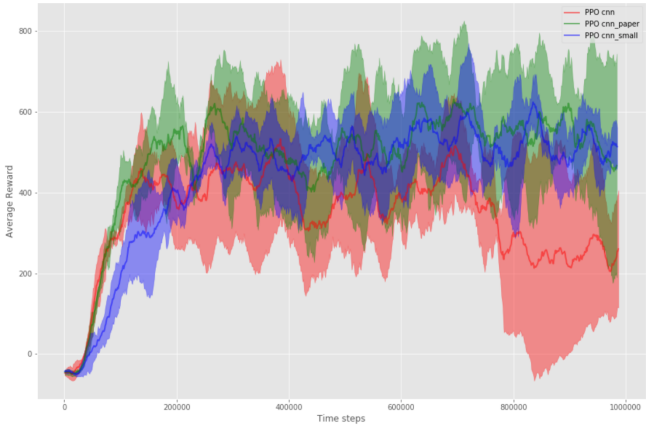

Fig. 2: Average reward of PPO with CNN vs. CNN paper vs. CNN small over training 1e6 timesteps (n=4).

B. Training in CARLA

We trained the agent in CARLA using the PPO implementation of the algorithm provided by Stable Baselines [19]. The training using the reward function from equation 5 was performed in the following setup:
Each training was conducted for 1e5 timesteps in Town 5. Within this town, we defined six different spawn positions on which the agent was spawned randomly during the training process. The spawn points are illustrated in Figure 8 in the Appendix. They are located on a highway, in a residential area, and on a normal road. Together they include straight lanes and curves with different lane markings such as solid white lanes as well as broken white lanes.
Furthermore, we defined three conditions under which a training episode will be ended. During training one episode terminates if the agent is involved in a collision (1) or if the number of timesteps exceeds 3000 (2). In further training configurations, we added the condition that the episode terminates in case the distance to the centerline exceeds 1.8 (3).

Analogous to CarRacing, we preprocessed the input images and varied the CNN network structures. Additionally, we increased the number of steps to run for each environment per update, incorporated frame skipping and further developed the reward function to enhance the agent's performance.

1) Network Structures: Regarding the CNN network structures, we used the same three networks as presented in IV-A.1.

2) Preprocessing: We adapted the observations the algorithm receives during training. Based on the experience from CarRacing, we firstly used low-resolution images of size

84x84 from the RGB and segmentation camera as shown in Figure 3. The camera was mounted in the front of the

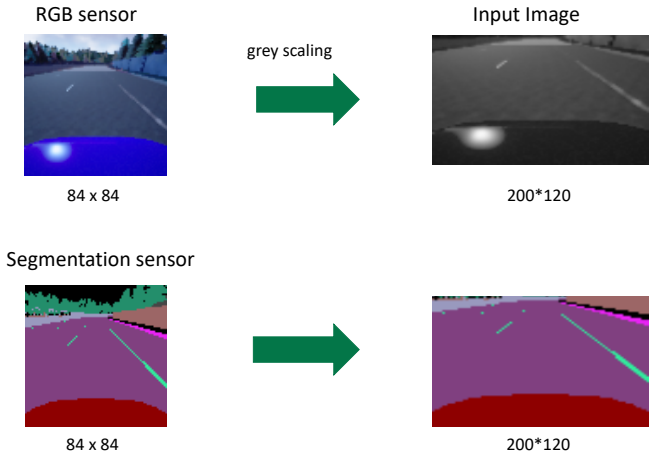

Fig. 3: Preprocessing of grey scale and segmentation images with low resolution.

car. To remove unnecessary information such as the sky and trees in the background, we resized the images to the size of 200x150 and cropped them to 200x120. In addition, we performed grey scaling on the images taken from the RGB camera.

After training with these two different input images respectively, the segmentation input image led to a better result, which we elaborate in section V-B. Thus, further improvements were only conducted on the segmentation image. Since the input image was blurred impeding the recognition of the lane markings, we changed the resolution of the segmentation camera to 1024x720. This provides a high-quality source for the features of different lanes. We resized and cropped the image to 200x100 pixels. However, the hood of the car occupies a large area of the lower part of the image. Therefore, we changed the pitch angle of the camera from -30 to -10 to raise the camera up. This procedure is displayed in Figure 4. Changing the pitch angle

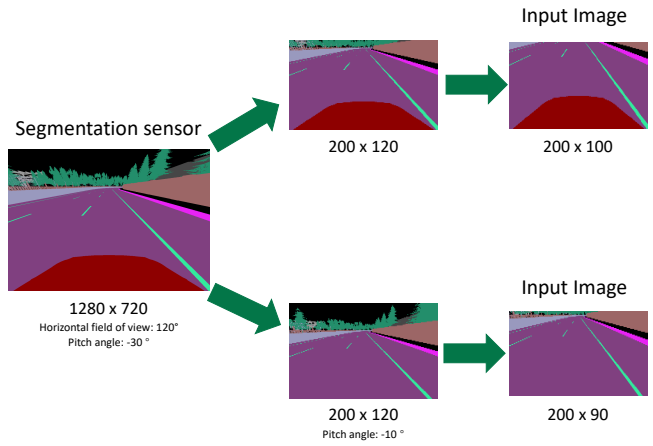

Fig. 4: Preprocessing of segmentation images with high resolution.

removes the car's hood, however, it adds more irrelevant information at the top of the image. Thus, we cropped again resulting in an input image of 200x90 pixels.

3) Hyperparameter Tuning: Initially, we trained all model configurations with the default hyperparameters provided by Stable Baselines [19]. Moreover, we chose the best performing model configuration and increased the hyperparameter "n_steps" from 128 to 1024 in order to enable the agent to gain more experience before a policy update is conducted.

Configuration 1 in Figure 5 displays the best model's reward per episode over 1e5 timesteps of training with n_steps equals 128 and no frame skipping. Its configuration can be found in row 4 of Table II in Section V. When increasing n_steps to 1024 and frame skipping to two respectively four (see row 7 and 8 of Table II), configuration 2 and 3 in Figure 5 display a more stable training process over time and the reward per episode increased gradually.

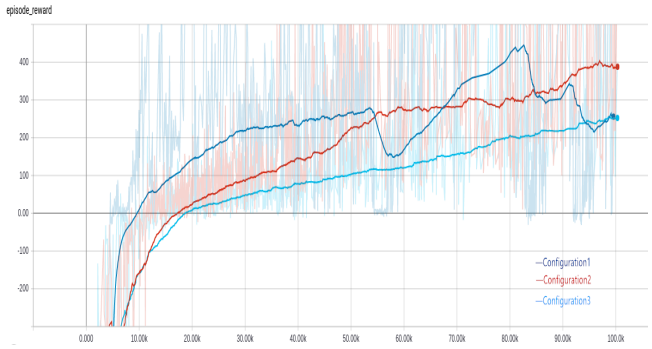

Fig. 5: The reward per episode for training configurations 1 - 3 for 1e5 timesteps. Configuration 1: row 4 of Table II with n_steps = 128 and no frame skipping, configuration 2: row 7 of Table II with n_steps = 1024 and frame skipping = 2, configuration 3: row 8 of Table II with n_steps = 1024 and frame skipping = 4.

Based on the insights gained during training we further developed the theoretical framework of the reward function (see equation 5) which we then evaluated qualitatively.

During training the car was speeding causing it to break out in turns easily. To tackle this unstable driving behavior, we removed the acceleration term (velocity difference $(v_t - v_{t-1})$) from the reward function. Additionally, we replaced the speed term (traveled distance $\|p_t - p_{t-1}\|_2$) by function 6. Thereby, the exceeding of a certain speed limit is punished whereas the acceleration up to the target speed is rewarded.

$$r_{speed} = \beta * 5 \|p_t - p_{t-1}\|_2 + (1-\beta) * (-5)(v_t - speedlimit) \quad (6)$$

$$with\ \beta = \begin{cases} 1, v_t \leq speed\ limit \\ 0, v_t > speed\ limit \end{cases}$$

Furthermore, the agent was lacking sensitivity towards weight changes during training. This problem is caused by the different codomains of the partial reward functions. Whereas the collision and intersection with lanes are binary

terms, the centerline reward can be any positive number. Thus, the impact of weights for binary rewards vanishes. To solve this problem, we normalized the reward function by replacing the term r_{speed} by the function:

$$f(x) = \begin{cases} \frac{x}{8}, x \leq 8 \\ \frac{8}{x}, x > 8. \end{cases} \quad (7)$$

$r_{centerline}$ was replaced by the function $f(x) = \frac{x}{1.8}, 0 \leq x \leq 1.8$. Both functions evoked the desired driving behavior and are plotted in Figure 9 in the Appendix.

Lastly, we trained the agent in an environment with other cars as well as static objects. Though collisions are weighted negatively in the reward function the agent did not take other objects into consideration and drove straight into them. Hence, we added the term $-7 * r_{closest\ vehicle}$ to the reward function with

$$r_{closest\ vehicle} = \begin{cases} 1, distance\ to\ closest\ vehicle < 2 \\ 0, distance\ to\ closest\ vehicle > 2 \end{cases} \quad (8)$$

Thereby, the reward is reduced if the distance to the closest vehicle goes below a threshold of 2 and a collision is punished before it actually happens.

V. Evaluation

A. Evaluation in CarRacing

In order to evaluate the different model configuration resulting from training the agent, we constructed the following scenario:

To obtain the average reward of a configuration we trained four models with the same configuration (n=4) and conducted 50 test runs in the CarRacing-v0 environment which we then averaged. The results are summarized in the Figure I. Due to time constraints, we neglected the CNN architecture "CNN paper". Whereas preprocessing increased the average reward of models using the default CNN, it decreased the average reward of models with small CNN. Also, we set the entropy coefficient to 0.01 for the best performing model. However, the increase in exploration negatively affected the average reward. Hence, the configuration which does not perform preprocessing nor exploration and uses the 'CNN small' architecture turned out to be the best performing model.

B. Evaluation in CARLA

After training a model configuration, we evaluated the trained model using the following scenario: We chose a spawning point in Town 5 different from the respective spawning points used for training. From this point, the agent starts driving until a collision occurs evoking respawning. Each model configuration was evaluated for 5 rounds each of them encompassing 2000 timesteps. Finally, we calculated the average reward per episode as well as the average reward per timestep to be able to assess the respective configuration. A comparison of the evaluation results is displayed in Table II. While comparing several models using grey scale and segmentation images as CNN network inputs while keeping the other parameters the same resulted in a better performance of the agent with the segmentation images as inputs. This case is illustrated exemplarily in the first two rows of Table II. Thus, in the course of optimizing the agent's performance, grey scale images were not taken into consideration.

While observing the driving behavior of the trained agents in rendering mode, we noticed that the agent often snakes in lines as it tries to stay within a designated lane which we refer to as "drunk driving" in the following. As the reward function includes a discriminating factor that penalizes moving away from a lane's centerline, one explanation for this behavior might be that every time the reward is updated the agents tries to minimize a penalization by steering in the opposite direction. However, as the agent is rewarded after its action the reaction is too late and therefore causes drunk driving. As the defined driving task only includes driving on roads with wide bends, we restricted the steering range from [-1, 1] to [-0.2, 0.2] leading to smoother driving behavior and an increase in average reward per episode.

As a next step, we changed the input images from low to high resolution and enlarged the horizontal field of view from 90 to 120. Additionally, a third terminal condition was added that respawns the agent once the distance to centerline exceeds 1.8 while training. These measures increased the agent's performance in terms of average reward per episode from 252 to 397.16. Increasing n_steps to 1024 and frame skipping to two respectively four resulted in a better generalization of the model when watching the agent drive in the evaluation scenario. Moreover, an average reward per episode of 290.56 respectively 252.52 which is similar to the one of the best performing model could be achieved. As this model configuration (see row 4 in Table II) led to the best agent's performance using the CNN network structure, we evaluated this configuration using CNN paper and CNN small. However, the results did not exceed the previous configuration.

Apart from these quantitative results, we evaluated qualitatively the adaption of the reward function as described in IV-B. We did so by observing the agent's driving behavior during training.

Removing the acceleration term and replacing the speed term proved to be successful measures to slow down the car. Just as desired the car accelerates until it meets the target speed and then keeps that speed. After normalizing the reward function the car reacts more sensitively to weight changes e.g. a high weight for $r_{centerline}$ causes the car to drive in the middle of the road, extremely slow though. In order to find the best weights, extensive training needs to be done. Although we added an additional term punishing short distances to other objects the car sill crashes frequently. This can be explained by the fact that there is not enough training data for such situations; thus, the policy is not updated accordingly.

C. Discussion and Limitations

Concerning the state space of CarRacing, we could conclude that a shallower convolutional neural network con-

	CNN		CNN paper	CNN small		
		Preprocessing 84x84			Preprocessing 84x84	Entropy Coefficient 0.01
Average Reward	515.09	586.52	433.93	787.65	529.98	612.98

TABLE I: CarRacing Evaluation.

Sensor image	Terminal Condition	Resolution	CNN structure	Camera view angle	Steering angle	Average reward per episode	Average reward per tilmestep
Grey scale	1 & 2	Low	CNN	90	[-1, 1]	-12.45	0.05
Segmentation	1 & 2	Low	CNN	90	[-1, 1]	-9.32	0.12
Segmentation	1 & 2	Low	CNN	90	[-0.2, 0.2]	252	0.56
Segmentation	1 & 2 & 3	High	CNN	120	[-0.2, 0.2]	397.16	1.84
Segmentation	1 & 2 & 3	High	CNN paper	120	[-0.2, 0.2]	52.78	0.92
Segmentation	1 & 2 & 3	High	CNN small	120	[-0.2, 0.2]	112.58	1.3
Segmentation	1 & 2 & 3	High	CNN	120	[-0.2, 0.2]	290.56*	2.13*
Segmentation	1 & 2 & 3	High	CNN	120	[-0.2, 0.2]	252.52**	1.33**

TABLE II: Evaluation in Carla. Each configuration was evaluated over 5 rounds for 2000 timesteps. The terminal conditions are: 1. Collision. 2. Timesteps $>$ 3000. 3. Distance to centerline $>$ 1.8. * n_steps = 1024, frame skipping = 2; ** n_steps = 1024, frame skipping = 4.

tributed to a better performance of the agent. This might be due to the simpler nature of the 96x96 RGB images which only display a grey race track and a green lawn. In contrast, the maps in CARLA contain a level of detail similar to the real world. The use of segmentation images made it possible to reduce this multitude of information and to display these information (e.g. lane markings) clearly in the images. In combination with higher resolution images and larger camera angles capturing more road markings, this has contributed to a significant improvement in the agent's performance. Additionally, in contrast to CarRacing, the features could be learned more accurately from deeper convolutional networks. With the reward function designed in III-C.4, we were able to train an agent that masters the first two tasks defined in III-C.1 in an empty map. Including other cars, pedestrians and obstacles to the map resulted in a significant increase in complexity which we started to address by further developing the reward function. However, even though we included a term for controlling the distance to other vehicles (see equation 8), there is still room for improvement. At this point, the difficulty arises to design a reward function that enables the agent to drive accident-free in complex driving situations and at the same time does not determine the agent's behavior completely. Here, a multiple-goal reinforcement learning approach as proposed by [20] can be a solution to handle such complex driving scenarios.

VI. Conclusion and Outlook

In this paper, we applied deep reinforcement learning with continuous control to address the autonomous driving task. First, we trained a Proximal Policy Optimization (PPO) agent in the CarRacing-v0 environment. To maximize the reward, the agent must drive through the racetrack as fast as possible without getting off the road. In order to optimize the agent's performance, we examined different image-based inputs, network structures, and hyperparameters. All models were assessed by training four models with the same configuration and then averaging the reward of 50 test runs. Finally, the best performing model achieved an average reward of 787 without performing preprocessing nor exploration using the "CNN small" architecture.
Based on our results, we transferred the PPO algorithm to the more complex CARLA environment. We trained the agent to stay within a designated lane on the road while taking turns along the way. Additionally, we experimented with more complex driving situations including the presence of static obstacles and other driving cars. Analogous to CarRacing, we preprocessed the input images and varied the CNN network structures. Moreover, we increased the number of steps to run for each environment per update, incorporated frame skipping and further developed the reward function. All model configurations were trained for 1e5 timesteps and subsequently evaluated for 5 rounds with 2000 timesteps per round. Ultimately, the best performing model reached an average reward per episode of 397. The underlying configuration of that model is the following: high-resolution segmentation images with an enlarged horizontal field as input, "CNN" as network architecture, restricted steering range as well as the application of all three terminal conditions. The resulting agent can stay within a traffic lane and take turns; however, its driving ability does not encompass appropriate interaction with other road users.

Future work should address the design of the partial reward functions as they essentially determine the agent's behavior and thus its performance. With regard to the input images, the amount of information extracted can be increased. Particularly, individual objects and their 3D dimensions need to be identified so that the agent can distinguish between individual objects. Furthermore, the agent's interaction with the environment should be improved by taking pedestrians, traffic lights, traffic signs, etc. into consideration. Lastly, the PPO agent should be trained longer with more varying spawning positions to allow for better generalization.

REFERENCES

[1] X. Liang, T. Wang, L. Yang, and E. Xing, "Cirl: Controllable imitative reinforcement learning for vision-based self-driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 584–599.

[2] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[3] 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.

[4] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[5] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[6] S. Jang, J. Min, and C. Lee, "Reinforcement car racing with a3c," 2017.

[7] 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)*. IEEE, 2017, pp. 244–250.

[8] C. Galias, A. Jakubowski, H. Michalewski, *et al.*, "Simulation-based reinforcement learning for autonomous driving," *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[9] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," *arXiv preprint arXiv:1904.09503*, 2019.

[10] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *arXiv preprint arXiv:1704.03952*, 2017.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[12] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

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

[14] 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 *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8248–8254.

[15] D. van der Wal, B. O. K. Intelligentie, and W. Shang, "Advantage actor-critic methods for carracing," 2018.

[16] M. A. F. Khan and O. H. Elibol, "Car racing using reinforcement learning," 2016.

[17] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[19] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[20] D. C. K. Ngai and N. H. C. Yung, "A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 509–522, 2011.

APPENDIX

Fig. 6: Input image for CNN network.

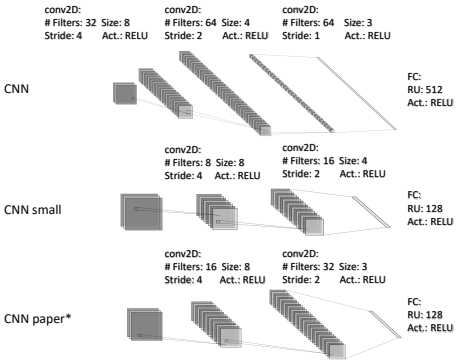

Fig. 7: CNN network structures. *[6].

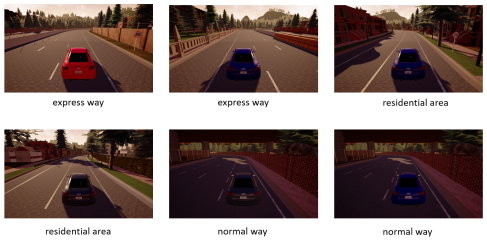

Fig. 8: Defined spawn points for training.

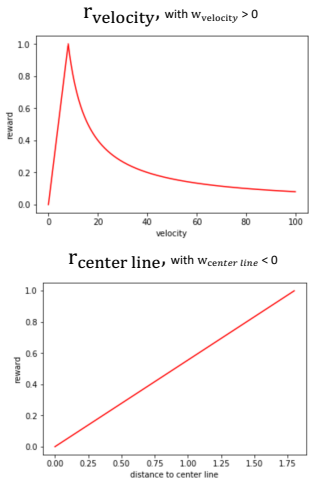

Fig. 9: Curve of normalized codomains of partial reward function.