Formula RL: Deep Reinforcement Learning for Autonomous Racing using Telemetry Data

Adrian Remonda[1], **Sarah Krebs**[1], **Eduardo Veas**[1], **Granit Luzhnica**[1], **Roman Kern**[1]
[1] Know-Center - Graz, Austria
{aremonda, skrebs, eveas, gluzhnica, rkern} @ know-center.at

Abstract

This paper explores the use of reinforcement learning (RL) models for autonomous racing. In contrast to passenger cars, where safety is the top priority, a racing car aims to minimize the lap-time. We frame the problem as a reinforcement learning task with a multidimensional input consisting of the vehicle telemetry, and a continuous action space. To find out which RL methods better solve the problem and whether the obtained models generalize to driving on unknown tracks, we put 10 variants of deep deterministic policy gradient (DDPG) to race in two experiments: i) studying how RL methods learn to drive a racing car and ii) studying how the learning scenario influences the capability of the models to generalize. Our studies show that models trained with RL are not only able to drive faster than the baseline open source handcrafted bots but also generalize to unknown tracks.

1 Introduction

Autonomous driving has received a lot of interest from the media and research alike, due to its potential to change how mobility and transport may look like in the future. However, within the domain of autonomous driving, there are different scenarios which dictate the objective for which to optimize. The focus of our research is to provide autonomous driving for racing cars in order to assist professional drivers to improve their racing line. In this case, the ultimate goal of our driver model is to drive the car around a racing track so as to achieve the lowest possible lap-time, preferably by reaching the physical limits of the car. In common practice, this problem is addressed by applying methods from the field of control theory. These methods utilize heuristics and demand domain knowledge to tune the model's parameters [Segers, 2014] manually. As each racing track has its peculiar challenges, these methods often require a different set of heuristics and parameters for each new situation.

Reinforcement learning aims at training an agent to learn to interact with an environment such as to maximize some notion of long-term reward. Combining RL with deep learning, problems with high-dimensional state spaces can be solved [Mnih *et al.*, 2013; 2015]. With algorithms like deep deterministic policy gradient (DDPG), deep RL can be extended to allow for solving continuous action space optimization problems [Lillicrap *et al.*, 2015]. This is an essential prerequisite for our use case as the racing car expects inherently continuous control (i.e. steering, brake, and throttle).

As regards autonomous driving, a lot of attention has been devoted to image processing, to analyze information based on images from cameras [Chen *et al.*, 2015; Koutník *et al.*, 2013; Drews *et al.*, 2017; Jaritz *et al.*, 2018]. In general, processing video input takes significant time and resources in training phases: from rendering to learning from them. Instead, a physics engine for racing simulation often runs headless to produce telemetry data at high rates (1000Hz), and racing cars already have the infrastructure to capture and make use of telemetry data which could be utilised for our purpose. Thus, our goal in this paper is to investigate how well different RL models can perform by using solely telemetry data streams. Besides, images contain a wealth of information which makes it difficult to extract the essentials needed to drive a racing car to its physical limits. For instance, there are many corners were the visibility is limited by obstacles. That is why, human pilots need to learn the track by heart before the race which helps them to decide how to manage (enter or leave) corners. Similarly, our algorithms use a (partially observable) racing line as reference for decision. It is common knowledge that besides vision and knowledge of the track, drivers heavily rely on the lateral and longitudinal acceleration that they feel through the vestibular system as well as the saturation of the tires by sensing the torque present in the steering wheel [Bentley, 1998]. Such information is well represented by the telemetry data of racing cars.

Given our goal of autonomous racing, this work is driven by two research questions, with the first being (**RQ1**): *is it feasible to learn a driver model that effectively drives a racing car by relying only on telemetry data as input? More concretely, which combination of algorithm/architecture is best suited to solve this task?*. Additionally, inspired by the common practice of human pilots, that train for the specific racing track (prior to the race), even though they already are proficient at driving, we explore the second research question (**RQ2**): *how well such algorithms trained on one track are able to generalize to other tracks?*

In the spirit of the competition, our experimental design is a tournament, where 10 variants of DDPG compete with each other in two studies. In the first study, 10 algorithms are trained to drive on a simple racing track. Algorithms producing the fastest models in terms of lap-time are promoted to a second part, to compare learning to drive on difficult tracks. In the second study the best models are tested on unknown tracks to assess how the current state-of-the-art in deep RL generalizes to unseen situations.

In our studies, RL models trained (from scratch) outperform the best performing open source bots available for our simulation environment. As a result, the faster models deliver a new racing line that leads to better performance. To the best of our knowledge, we are the first to explore racing line optimization problem using RL, which the main contribution of this work. Our further contributions to the modeling the autonomous racing problem using RL methods include: a simple scheme that reduces the exploration space when two outputs should be mutually exclusive in continuous action spaces and a modification of the target to solve an issue that arises when sampling end-of-episode transitions. Additionally, we propose using the look ahead curvature (LAC) which provides information regarding the upcoming shape of the track recorded in a previous lap. We also benchmark the models resulting from different algorithms on the trained and unseen environments (racing tracks). Our research clearly shows that these tests need to be considered in the racing domain.

2 Background

2.1 Autonomous Racing

Autonomous driving has matured into a field where reliable models are needed for obstacle detection and avoidance, maneuver initiation and recovery, while driving is reduced to path planning and path following. These models require a sophisticated perception of the environment, which is why much autonomous driving research focuses on image processing. The context of motorsports has its own requirements, that significantly influence the definition of the problem space. In contrast to passenger vehicles, where safety is the top priority, the main objective of a racing vehicle is to minimize the lap-time. One way to do this is to find, within the boundaries of the track, the trajectory where the car can move with the best lap-time–*the optimal racing line*. The optimal racing line is the best compromise between the shortest path and the trajectory that allows to achieve the highest speeds [Braghin *et al.*, 2008]. It depends on several factors including the track shape, the car aerodynamics, grip, etc. [Cardamone *et al.*, 2010]. Hence, the problem of trajectory planning is a bounded optimization problem that requires to take into account not only the geometry of the track but also vehicle dynamics. Autonomous racing thus requires a perception of the vehicle dynamics in relation with the environment. Typically, the optimal racing line is calculated offline or is estimated by the reference of an expert human driver.

Attempts to achieve the lowest possible lap-times with autonomous racing cars typically combine control theory, determining and utilizing the optimal racing line, and/or optimizing the driver model directly. [Cardamone *et al.*, 2010] calculated the ideal racing line using a genetic algorithm and then measured the lap-time of a line-follower bot. Their method outperformed the previous state-of-the-art models by a small margin while ours outperformed the state-of-the-art by a large margin and without having the necessity of having a line follower bot and just a reference line (this is typically done in professional racing scenarios, where a loose racing line exists that racers have as a reference). Another disadvantage of their method, is that it is unable to generalize to different tracks and limited to the performance of the bot. The racing line needs to be re-calculated for every new track or new car.

The aim of [Koutník *et al.*, 2013] was to demonstrate that a car can be autonomously driven by using images. Their approach performed a prepossessing step in order to reduce the state space by representing it in the frequency domain, converting the images in a set of coefficients that are then transformed into weight matrices via an inverse Fourier-type transform. However, their approach targets driving in general and not racing which would require to minimise the lap time. In contrary, we aim to reduce the lap time which is the most important factor in racing. [Ahmad El Sallab, 2017] tackle the generalization issues in traditional imitation learning this means that they need to have demonstrations in all tracks. This demonstrations came from a traditional proportional-integral-derivative controller (PID controller) with access to the position of the ego car with respect to the left and right lanes. All the previously mentioned papers try to follow the center of the track, which makes it impossible to achieve the optimal racing line. We are using as a loose hint a racing line that we then improve by a large margin. [Koutník *et al.*, 2013], [Jaritz *et al.*, 2018] used discrete action space while we are using continuous actions space. Our preliminary studies (not included in this paper) showed that for a one-dimensional action space (steering wheel), a discrete actions space algorithm such as DQN might learn a policy that is able to drive on the track, but is far from reaching the lap-times achieved by the continuous actions space algorithm DDPG (limited to steering wheel control as well). In our opinion,it is not possible to drive to the limits by discretizing the throttle and steer. In comparison, human gamers of racing games/sims consider a dedicated steering wheel a worthwhile investment. In professional simulators, as the ones used to train F1 pilots, it is a must. [Drews *et al.*, 2017] present a deep learning method to generate cost maps learned from human demonstrations. The cost map is then feed to a model predictive control algorithm (MPC) that runs in real time on a real 1:5 scale autonomous vehicle by sampling trajectories using a model of the dynamics of the car. The dynamics model was learned directly from the data. Although, this work represents the state-of-the-art in the field of control theory and it has several advantages it still needs labeled data that would have to be recorded for each different track. It also has the drawback that the driving performance is limited by the quality of the human demonstrations.

2.2 Motorsports

As above introduced, the optimal racing line depends on factors including the track shape, the car aerodynamics, the grip, etc. [Cardamone *et al.*, 2010]. It is often calculated offline or estimated by the reference of an expert human driver.

Racing line. The racing line is defined as a sequence of points on the track. Each point P_i of the racing line can be represented by the pair $\langle \delta_i, \alpha_i \rangle$. As depicted in Figure 1, α_i is

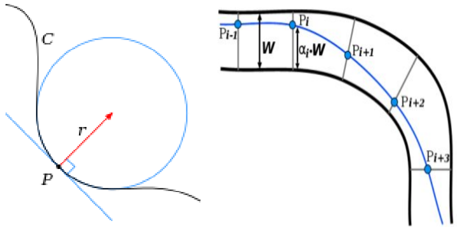

Figure 1: *Left:* Geometrical representation of the radious of curvature *Right:* Racing line representation [Cardamone *et al.*, 2010]

the lateral distance of the point from one of the track borders (e.g., from the right border), normalized by the track width W, and δ_i the distance of the point from the track starting line, computed along the track axis.

Figure 1 shows the **curvature** κ of the curve C at a given point P in track is defined by the inverse of the curvature radius (r) at that point, and it's given by $\kappa = \frac{\partial\theta}{\partial s}$ where κ is the curvature at a segment ∂s with a change of angle $\partial\theta$[1]. The smaller the curvature, the higher the speed that a car can maintain along the racing line. The maximum possible speed without loosing grip is given by $v_{max} = \sqrt{\mu\rho(g + F_a/m)}$ where m is the car's mass, μ is the tire-road friction coefficient, ρ is the curvature radius and F_a is the downforce. The shortest path is the sequence of points in one lap that results on the shortest distance, while the minimum curvature path is the sequence of points that allows to complete one lap with the minimum possible curvature. [Cardamone *et al.*, 2010].

2.3 Reinforcement Learning

Reinforcement learning deals with the problem of learning optimal behaviours for the interaction of an agent with an environment by trial and error, such as to maximize the accumulated reward obtained from the environment. It is assumed that the interaction of the agent with the environment takes place in discrete time steps t. At each step, starting from a state s_t, the agent executes an action a_t and receives a reward r_t and a new state s_{t+1} from the environment. The return from a state is defined as the sum of discounted future reward, $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r_i$, where $\gamma \in (0, 1]$ is a discount factor and T is a terminal time step after which the process restarts. The objective of RL is to learn a policy π, mapping states to actions, that maximizes the return from the start distribution. There are two main approaches for solving RL problems: methods based on value functions and policy search. So called actor-critic approaches employ both value functions and policy search. A policy π defines the agent's behavior by mapping states to actions. A value function provides an estimation of the future return and thus can be used to evaluate how good an action or state is. An action value function $Q^\pi(a_t, s_t) = \mathbb{E}_\pi[R_t|s_t, a_t]$ estimates the return starting from state s_t, taking action a_t, and then following policy π.

In deep RL the algorithm components are implemented as deep neural networks. The first successful deep RL algorithm was deep Q-network (DQN)[Mnih *et al.*, 2013], which succeeded at solving problems with high-dimensional state spaces (e.g. pixels), but can only handle discrete, low-dimensional action spaces (e.g. left, right). Driving a car requires continuous actions (steering, throttle). The algorithms thereto are topic of the next section.

3 Algorithms

3.1 Deep Deterministic Policy Gradient (DDPG)

By combining the insights of DQN with the actor-critic deterministic policy gradient algorithm, DDPG [Lillicrap *et al.*, 2015] allows for solving a wide variety of continuous control tasks. DDPG utilizes an actor function $\mu(s|\theta^\mu)$, specifying the current policy, and a critic function $Q(s, a|\theta^Q)$, both approximated by neural networks. At each step, based on the current state s_t, the agent chooses an action according to $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$, with a noise process $\mathcal{N}$ to allow for exploration, and obtains a reward r_t and a new state s_{t+1} from the environment. The observed transitions (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer. At each step, a minibatch of N transitions is uniformly sampled from the buffer. The parameters of the critic network are then optimized using Adam optimization to minimize the loss given as:

$$L(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{1}$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{2}$$

where y_i is the one-step target with the discount factor γ. Here, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ are the target networks associated with $Q(s, a|\theta^Q)$ and $\mu(s|\theta^\mu)$. Their parameters are updated at each step using soft updates, i.e. $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. To update the parameters of the actor network, a step proportional to the sampled gradient of the critic network with respect to the action is taken, which is given by:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_i}. \tag{3}$$

[Lillicrap *et al.*, 2015] evaluated DDPG on a car racing problem. They reported that, for both using low-dimensional data and using pixels as input, some replicas learned reasonable policies, while others did not. An open source implementation of DDPG [2] replicates those results to learn steering, or break. Instead, we introduce numerous modifications to the algorithm that make it possible to learn reasonable policies for racing and outperforming well-known baselines.

Window sampling. In partially observable environments, accessing a single state does not reveal the full underlying state of the environment at each time step. Window sampling provides the agent additional information by feeding a window of the last w states to the actor and the critic network.

Long short term memory. Another natural approach to include knowledge from past experiences is to make use of recurrent neural networks, which are able to remember information for an arbitrary number of time steps [Hochreiter and Schmidhuber, 1997]. For example, LSTM units can be added to the actor and the critic network.

Multi-step targets. For updating the critic function, a one-step target is used in standard DDPG. Multi-step targets [Mnih *et al.*, 2016; Vecerik *et al.*, 2017] incorporate the next

[1] https://en.wikipedia.org/wiki/Curvature

[2] https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html

n rewards obtained along the trajectory starting from state s_t and following a policy close to the current policy $\mu(s|\theta^{\mu})$ at time step t. The one-step target y_i (in Eq. 2) is replaced by:

$$y_i^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{i+k} + \gamma^n Q'(s_{i+n}, \mu'(s_{i+n}|\theta^{\mu'})|\theta^{Q'}) \quad (4)$$

Prioritized experience replay. In standard DDPG, transitions are sampled uniformly from the replay buffer at each step. Prioritized experience replay (PER) [Schaul *et al.*, 2015] attempts to make learning more efficient by sampling more frequently transitions that are more important for learning. The probability of sampling a particular transition i from the replay buffer is given by $P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}$, where p_i is the transition's priority. The sum in the denominator runs over all transitions in the buffer. Similar to the implementation outlined in [Vecerik *et al.*, 2017], we use $p_i = \delta_i^2 + \lambda_3 |\nabla_a Q(s_i, a_i|\theta^Q)|^2 + \epsilon$. Here, δ_i is the temporal-difference error calculated for the transition when it was sampled the last time, the second term represents the transition's contribution to the actor loss, λ_3 is used to weight the two contributions, and ϵ is a small positive constant ensuring all transitions are sampled with some probability.

3.2 Extensions of RL Algorithms

To ensure that DDPG variants work on racing tracks, we introduced: a method to reduce the exploration of continuous action spaces with two mutually exclusive outputs, a modification of the training objective for episode terminations, and a variation of the critic network.

Brake exploration The most important actions to control a racing car are steering, accelerating, and braking. Learning how to steer is rather straightforward, but the complex interplay between brake and throttle is very challenging from the exploration perspective. Following [Lillicrap *et al.*, 2015], we use an Ornstein-Uhlenbeck (OU) process [Uhlenbeck and Ornstein, 1930], which outputs temporally correlated values centered around a given mean, for noise generation. This allows for temporally correlated exploration. The values generated by the OU process are attenuated proportional to a parameter ϵ', where ϵ' is set to 1.0 in the beginning and is annealed to zero at the end of exploration phase. We add noise to three action dimensions independently, but with a probability of 0.1. We stochastically add a stronger noise to the brake while simultaneously lowering throttle by a factor of $(1 - \epsilon')$. This guides the exploration to not press the throttle and brake simultaneously, reducing the exploration space drastically.

Episode termination Episodes are terminated when a certain number of steps is reached, the car is out of track, moves backwards, or its progress along its longitudinal axis is slow. In all these cases except for the last, the agent receives a reward of -1 at termination. In all cases, the target (Equation 2) for the end-of-episode transition is replaced by $y_i = r_i$. The problem with this approach is that when terminating an episode due to reaching the maximum number of steps, the target function would also be $y_i = r_i$ for this end-of-episode transition, despite it being a good episode termination. Instead, to prevent an unintended adaption of the weights, in this case we use the same target (Equation 2) as for non-terminal transitions.

$$y_i = \begin{cases} r_i & \textit{premature end of episode} \\ r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) & i = max\_steps \textbf{ or } normal\ step \end{cases}$$

LSTM critic network Initially, we added LSTM units to both the actor and the critic network, but this lead to an unstable behavior of the actor. Thus, following the DQN approach for discrete action spaces in [Hausknecht and Stone, 2015], we kept the LSTM only in the critic. This is expected to improve the performance of the algorithm, as a good approximation of the Q value function is the basis for learning a good policy. The LSTM layer is placed after the concatenation of the state and the action stream. The input to the critic is a window of the last w states and actions.

4 Experiments

We evaluate the different algorithms on the open-source simulator TORCS (The Open Racing Car Simulator) [Wymann *et al.*, 2014]. TORCS is used for research in RL and autonomous driving, utilizing either images or low-dimensional features as input, e.g. [Lillicrap *et al.*, 2015]. At each time step, the agent receives detailed information about the state of the environment. However, as many parts of the simulation are not directly accessible to the agent, the environment is partially observable even if only a single car is on the track. The interaction with the environment takes place in discrete time steps with a spacing of 200 ms. The input to the algorithms consists of vehicle telemetry data. We carefully selected the features in Table 1. The reward is calculated as:

$$r = V_x(\cos(\theta) - \sin(\theta) - |\text{distance to track axis}|) \quad (5)$$

where θ is the angle between the car direction and racing line.

We compare three approaches: i) the middle line of the track is considered to be the track axis, which is unlikely to lead to an optimal driving behavior, as the middle of the track is not the optimal racing line. ii) a racing line from the Tita bot. The algorithm optimizes its path taking the racing line as a loose reference which then it improves by generating a new. This reduces the exploration space significantly. iii) same as ii) but adding the LAC to the input state s_t.

Damage is a quantity calculated by TORCS each time the car hits a wall and is proportional with the impact of the hit on the car. We consider the negative of the damage magnitude in the reward function and we report the damage of a model as the cumulative sum over the entire testing phase.

The experiments use three tracks considered by [Cardamone *et al.*, 2010] with increasing complexity (see Table 1). The Michigan Speedway is a semi-oval track that can be driven without using the brake. Forza and Aalborg are far more complex and cannot be driven without braking. Especially Aalborg is rather technical, with sharp turns that have to be taken at lower speed as well as fast segments.

4.1 Algorithms

We considered four algorithms: DDPG, LSTM, MS and PER and each of them was tested with $2 - 3$ variation of hyperparamaters. We used four variations of DDPG: *WIN1, WIN4, WIN8* use window sampling with window sizes 1, 4, and 8

Notation	Description
θ	Angle between the car direction and the direction of the track axis or racing line.
track	Vector of 19 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 m.
trackPos	Distance between the car and the track axis or racing line.
V_x	Speed of the car along its longitudinal axis.
V_y	Speed of the car along its transverse axis.
V_z	Speed of the car along its z-axis.
$\vec{\omega}$	Vector of 4 sensors representing the rotation speed of the wheels.
f_{rot}	Number of rotations per minute of the car engine.
LAC	Look ahead curvature. Vector of 4 curvature measurements from the racing line at 20, 40, 60 and 80 meters ahead. The curvature is recorded from a previous slow lap.

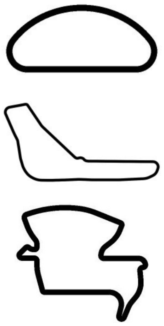

Table 1: *Left:* Telemetry features used as input [Loiacono *et al.*, 2013]. *Right:* Tracks used for evaluation [Wymann *et al.*, 2014]: Michigan (Top), Forza (Middle), Aalborg (Bottom). Note that LAC is not used in the 1st and 2nd part of Study 1.

(WIN1 is standard DDPG). *LSTM4, LSTM8* utilize an LSTM critic network with window sizes 4 and 8. *MS2, MS3, MS4* uses multi-step targets with 2, 3, and 4 steps. *PER40k, PER1M* utilize PER with buffer sizes 4×10^4 and 10^6.

4.2 Study 1: Learning to Drive

This study addresses RQ1 by examining whether RL models can drive a racing car from telemetry data and then comparatively evaluating the performance across different models.

Procedure. This study was split in three parts. To reduce the training time for hyperparameter selection, part1 uses a simple track, Michigan and trained for 500 episodes. The learned model was *tested* without exploration ($\epsilon = 0$) on the same track the results were used to select the hyperparametrs for each of the four used algorithms. In part2, the four algorithms (with selected hyperparameters) and tested on a technically more complex track, Aalborg, using both MOT and Tita bot as racing line references. With the selected best algorithm of part2, in part3 we evaluated the impact of adding future information of the track. We did so by adding the look ahead curvature (LAC) and training in all three tracks. Part 2 and 3 were trained for 7000 episodes. The results were used to select the best algorithm/model suited for the task of autonomous racing.

Results. Lap time is the most important measure in racing and we will utilise it to compare the performance of different models. Table 2 shows best (bLT) and average (aLT) lap-times in both Michigan and Aalborg as well as baseline approaches: Tita (heuristic state-of-the-art bot) and Genetic [Cardamone *et al.*, 2010]. The results (aLT) of Michigan track were used to select the best hyperparameters for each algorithm. Thus in turn, WIN4, MS4, PER1M and LSTM4 were selected to be used for Aarlborg track. As shown in Table 2, our models outperform the baseline bots by a large margin (bLT: WIN1 26.75 vs Tita 28.57) on the simple track. On the complex track, only models trained with racing line (RC) are faster (LT: PER1M 67.17 vs Tita 68.11). The results also show that PER1M trained with racing line achieves the best lap-time and thus it is selected to compare the effect of adding the LAC. Table 3 shows the results of adding the LAC to the state (PER1M). This addition gives the best result.

4.3 Study 2: Driving in New Scenarios

Study 2 on *driving in new scenarios* addresses RQ2 and investigates how the driver models perform in unseen tracks.

	Part I: Michigan		Part II: Aarlborg			
	MOT Reference				RC Reference	
	bLT	aLT	bLT	aLT	bLT	aLT
WIN1	**26.75**	27.16	-	-	-	-
WIN4	26.77	**26.96**	74.75	75.73	67.56	**69.89**
WIN8	26.75	35.61	-	-	-	-
MS2	26.79	30.33	-	-	-	-
MS3	26.80	27.13	-	-	-	-
MS4	26.83	27.04	77.68	78.53	69.94	77.01
PER40k	26.84	34.85	-	-	-	-
PER1M	27.06	32.30	71.76	75.22	**67.17**	70.11
LSTM4	27.35	27.52	85.87	90.05	-	-
LSTM8	27.50	27.85	-	-	-	-
Tita	28.57	-	-	-	68.11	-
Genetic	33.86	-	-	-	69.92	-

Table 2: Testing results over 10 runs in Michigan and over 5 runs in Aalborg. Best lap-time (**bLT**), average best lap-time (**aLT**) (in seconds) for models with 0 damage. The Aalborg track was trained/tested using both middle of the line (**MOT**) and racing line from Tita (**RC**) bot as racing line references.

Procedure. For this study, we use different variants of PER1M (MOT, RC, RC+LAC) as shown in Table 4. The models are trained in one track (Michigan or Aalborg) and tested in two other unseen tracks. While training, as the model improves, a test is performed in the other two unseen tracks. We choose as final model, referred to as *general model*, the one that performs with the best lap time in the training track but that is also able to finish the unseen tracks.

Results. First, the models trained in Michigan did not finish the unseen tracks (Aalborg and Forza). We attribute this to the simplicity of the Michigan track (no sharp corners) offering little exposure to complex manoeuvres and leading quickly to overfitting. Table 4 shows the results of the general models trained in Aalborg and tested in unseen tracks. These models were able to drive in unseen tracks. As expected, their performance in unseen tracks compared to the performance of models trained and tested in the same track (see Table 3) is significantly worse. Figure 2 illustrates lap-times achieved on each track for all PER1M models learned on Aalborg. The vertical red line indicates the *general* model. With more training, the performance improvements in Forza (test) are consistent with those in Aalborg (training), i.e., the more the model trained in Aalborg, the better it performed in Forza. But, these improvements were not reflected in the Michigan track, which we think is due to the very basic shape of the Michigan track (no steep curves).

	Forza	Michigan	Aalborg
Tita	77.39	28.57	68.11
PER1M MOT	-	27.060	71.758
PER1M RC Reference	75.306	26.906	67.27
PER1M RC Reference + LAC	**73.018**	**26.278**	**63.35**

Table 3: Lap times (best) of a comparison between the best models of the considered different approaches in Study 1. **Tita** vs **PER1M** with the reward function set to follow the middle of the track (**MOT**) vs bot racing line vs bot racing line + look ahead curvature (**LAC**).

	Best		General		
	bLT	aLT	Aalborg	Michigan	Forza
PER1M MOT	71.76	75.22	74.84	34.65	103.09
PER1M RC Ref.	67.17	70.11	71.26	33.67	108.69
PER1M RC Ref. + LAC	63.35	65.14	70.616	34.07	107.15

Table 4: Fastest models trained on Aalborg compared with *general* models that complete all tracks (Testing). DNF: no model finished.

5 Impact of the Contributions

5.1 Brake Exploration

Figure 3 depicts the outputs of a model trained with the brake exploration scheme proposed in Section 3.2. Using this exploration scheme, the model is able to press the throttle while completely releasing the brake and vice versa. This shows that the approach is capable of reducing the exploration space. We also observe that after braking and releasing the pedal, the model waits for some time until it starts steering. This is a common practice among professional racing drivers to avoid over-steering. The algorithm learned this behaviour by itself.

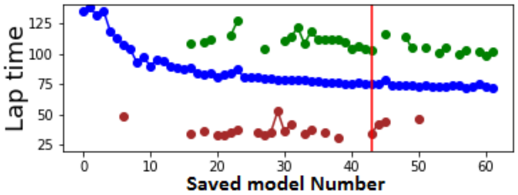

Figure 2: Fastest laps for all PER1M models trained on Aalborg and tested on Aalborg (Blue), Michigan (Brown), and Forza (Green).

5.2 Episode Termination

We compared two different settings for good episode terminations (terminations caused by reaching the maximum number of steps). First, we set the target to $y_i = r_i$ for the corresponding end-of-episode transitions (variant 1). Second, we set the target to $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ as for non-terminal transitions (variant 2) and we refer to it as adopted target (**AT**). Figure 4 shows the training performance in terms of average episode reward for the WIN1 algorithm for both variants when learning to control steering and pedal on the Michigan track. It can be seen that a higher maximum is reached for variant 2. This small modification is essential, especially when episodes are terminated after only a few steps as is done when training to optimize single corners.

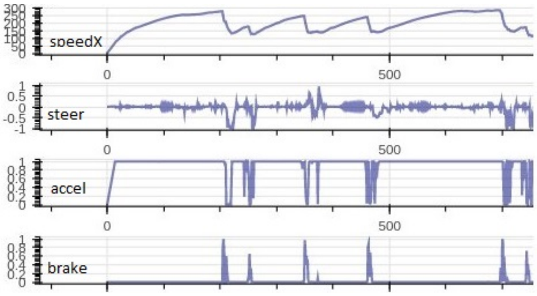

Figure 3: Outputs (steer, acceleration, brake) of a model trained with brake exploration and speed along the car's longitudinal axis.

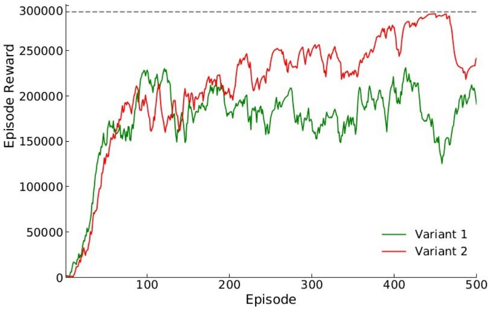

Figure 4: Training episode reward for normal end-of-episode (green) and **AT** (red) transitions. The reward is averaged over 10 training runs with different random seeds, smoothed with a moving average over 5 episodes.**WIN1** with episode terminations caused by reaching the maximum number of steps

6 Discussion and Outlook

We presented a comprehensive study applying RL methods to self-driving racing cars relying only on car telemetry data. Our goal was to investigate the capabilities of RL to drive from telemetry data, because simulated environments can produce telemetry data at high rates, which makes the use case practical. Besides, the telemetry data describes the physics vehicle dynamics, which is important since we intended to drive the car at its physical limits.

The results of Study 1 supports *RQ1*: Deep RL algorithms effectively learn to drive fast from telemetry data and obtain models better than state-of-the-art handcrafted models. The results also showed that PER1M (prioritized experience replay with a replay buffer size of 1M samples) was the best performing algorithm in a complex track. Most importantly, the results evidenced that our proposed look ahead of the curve approach (LAC) improves the performance of models in the self racing scenarios. To the best of our knowledge, these results constitute a contribution to the line optimization problem using reinforcement learning, and our self-thought models outperform the baseline open-source bots. Our findings also show that the resulting models are able to work with any racing line, making them suitable for street cars where the problem is typically to follow a given trajectory. Note that contrary to bots, our solution does not blindly follow a given racing line, instead, it takes it as loose guide and then it improves it to generate a new optimal racing line.

Second, we studied how the learned models perform on unseen tracks. In Study 2, we trained models on a simple and difficult track and compared their ability to drive on unknown tracks. The results show that models trained in reasonably complex tracks (Aalborg) can generalise relatively well in unseen (untrained) simple (Michigan) or more complex tracks (Forza). However, such models underperformed in terms of lap time compared with models trained and tested on the same track. It is interesting that such performance behaviour resembles the behaviour of professional human pilot drivers. Even though they are experts in driving and have acquired a lot of driving skills over the years, expert drivers still practice a lot for the competing track before the competition. This way they can memorise the landmarks and car dynamics before achieving their full performance.

Similar to human pilots, models trained in Aalborg in Study 2, did acquire driving skills, but in order to achieve the best performance, they would need to continue training for each specific track. Also, similar to human pilots that learn the track by heart, when assisting the model with the information about the curvature using our proposed look ahead of curve (LAC) method, the performance of the model improves.

Thus, in the future work, we will put more effort and further investigate means of generating such a general model which would achieve relatively high performance in unseen tracks and see how further training it for specific tracks, will affect the performance. This would improve training time. One way to make such general models robust and learn faster would be to use pre-defined maneuvers to train the model. Additionally, we plan to investigate whether we could train models without using any reference line during the training. In this work, we utilised TORCS as a simulation environment which was sufficient to evaluate our hypotheses and to demonstrate the capabilities of our approach, while still being fast to train/test and iterate. Yet, professional telemetry systems provide in real-time a wealth of information otherwise inaccessible, such as engine temperature variations, damper displacement, tire saturation. In future work, we intend to move towards golden industry standards in simulated environments and make use of more sophisticated telemetry information. Finally, we intend to investigate the interactions with using images as a baseline for comparison but more importantly as complementary channels.

Acknowledgement

This research was partially funded by AVL GmbH and Know-Center GmbH. Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry of Economy, Family and Youth and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

References

[Ahmad El Sallab, 2017] Omar Abdel Tawab Mohammed Abdou Ahmad El Sallab, Mahmoud Saeed. Meta learning framework for automated driving. 2017.

[Bentley, 1998] Ross Bentley. Speed secrets: Professional race driving techniques. 1998.

[Braghin *et al.*, 2008] Francesco Braghin, Federico Cheli, Stefano Melzi, and Edoardo Sabbioni. Race driver model. *Computers & Structures*, 86(13):1503 – 1516, 2008.

[Cardamone *et al.*, 2010] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *2010 IEEE Conference on Computational Intelligence and Games*, 2010.

[Chen *et al.*, 2015] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. DeepDriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[Drews *et al.*, 2017] Paul Drews, Brian Goldfain, Grady Williams, and Evangelos A. Theodorou. Aggressive deep driving: Model predictive control with a cnn cost model. 2017.

[Hausknecht and Stone, 2015] Matthew J. Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. *CoRR*, abs/1507.06527, 2015.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comp.*, 1997.

[Jaritz *et al.*, 2018] Maximilian Jaritz, Raoul de Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. 2018.

[Koutník *et al.*, 2013] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based torcs. 2013.

[Lillicrap *et al.*, 2015] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[Loiacono *et al.*, 2013] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *CoRR*, abs/1304.1672, 2013.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on International Conference on Machine Learning*, 2016.

[Schaul *et al.*, 2015] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, 2015.

[Segers, 2014] Jorge Segers. *Analysis Techniques for Racecar Data Acquisition*. SAE International, 2014.

[Uhlenbeck and Ornstein, 1930] George Uhlenbeck and Leonard Ornstein. On the theory of the brownian motion. 1930.

[Vecerik *et al.*, 2017] Matej Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR*, abs/1707.08817, 2017.

[Wymann *et al.*, 2014] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. http://www.torcs.org, 2014.