IMPLEMENTATION DETAILS

A. GTS Access

The GTS simulator runs on a *PlayStation 4*, while our agent runs on a separate desktop computer. We do not have direct access to the GTS simulator, neither do we have insights into the car dynamics modeled by GTS. Instead, we interact with GTS over a dedicated API via ethernet connection. The API provides the current state of up to 20 simulated cars and accepts car control commands, which are active until the next command is received. Whereas previous work in the area of RL often ran thousands of simulations in parallel to collect training data [1], [2], [3], we can only run one simulation per *PlayStation*, and for this work we make use of only 4 *PlayStations* during training. Moreover, the simulation runs in real-time and cannot be sped up for data collection during training nor can it be paused to create additional time for decision making. The simulator's state is updated at a frequency of 60 Hz, but, to reduce the load on the *PlayStations* when controlling 20 cars, we limit the command frequency to 10 Hz during training. During evaluation, we switch to one car, increasing the agent's action frequency to 60 Hz. We use a standard desktop computer with a *i7-8700* processor with a clock speed of 3.20GHz for inference and a *GeForce GTX 1080 Ti* graphics card for backpropagating through the models.

B. Network Training

To train the networks we roll out 4x20 trajectories per epoch in parallel, with a length of 100 seconds each, by using all 20 cars available on each of the 4 *Playstations*. To minimize disturbance between cars during training, we initialize the position of agents equally distributed over the racing track with an initial speed of 100 km/h, which we found can accelerate training, since it allows the agents to faster approach the maximal feasible segment speeds, which are of interest for finding the fastest possible trajectory.

For the training process we make use of the TensorFlow-based SAC implementation by OpenAI[1]. We modified the code base to allow to asynchronously learn during roll outs and changed the default 1-step TD error to a 5-step TD error to stabilize training as also used in [4]. The combined training of all 4 networks takes approximately 35 seconds per epoch, which leaves the total epoch time at 100 seconds. Because of the real-time character of the GTS environment, an extensive hyperparameter search is not feasible. We therefore adapt most default hyperparameters from the OpenAI implementation, except for those listed in table I.

Hyperparameter	Value
Mini-batch size	4,096
Replay buffer size	4×10^6
Learning rate	3×10^{-4}
Update steps per epoch	5,120
Reward scale ($1/\alpha$)	100
Exponential discount (γ) for the "Mazda Demio"	0.98
Exponential discount (γ) for the "Audi TT Cup"	0.982
Wall contact penalty scale (c_w)	5×10^{-4}
Number of range finders (M)	13 (every 15°)
Number of curvature mesurements (N)	10 (every 0.2s)

TABLE I
HYPERPARAMETERS USED FOR THE SAC ALGORITHM AND THE GTS SIMULATION. DENSER RANGE FINDERS LEAD TO SLOWER CONVERGENCE BUT SHOWED NO ADDITIONAL IMPROVEMENTS IN THE LEARNED POLICY.

C. Curvature construction

We base the input feature selection on the results of a pre-study on GTS conducted with behavioral cloning, aiming to learn human-like driving based on a regression of expert's actions on visited states. We iterated over combinations of subsets of features provided by GTS as well as additionally constructed features, which lead to the following feature selection: 1) The linear velocity $\boldsymbol{v}_t \in \mathbb{R}^3$ and the linear acceleration $\dot{\boldsymbol{v}}_t \in \mathbb{R}^3$. 2) The Euler angle $\theta_t \in (-\pi, \pi]$ between the 2D vector that defines the agent's rotation in the horizontal plane and the unit tangent vector that is tangent to the centerline at the projection point. This angle is the only direct way for the policy network to detect if a car is facing the wrong direction. 3) Distance measurements $\boldsymbol{d}_t \in \mathbb{R}^M$ of M rangefinders that measure the distance from the vehicle's center point to the M edge points of its surrounding objects, such as the edge of the race track. The rangefinders are equally distributed in the front 180° of the car's view. 4) The previous steering command δ_{t-1}. 5) A binary flag with $w_t = 1$ indicating wall contact and 6) N sampled curvature measurement of the course centerline in the near future $\boldsymbol{c}_t \in \mathbb{R}^N$. The curvature is represented through an interpolation of the inverse radii of circles fitted through centerline points provided by GTS.We therefore represent the observation at a given time step t as a vector denoted as $\boldsymbol{s}_t = [\boldsymbol{v}_t, \dot{\boldsymbol{v}}_t, \theta_t, \boldsymbol{d}_t, \delta_{t-1}, w_t, \boldsymbol{c}_t]$. To allow for a fair comparison with human drivers, we only use features that humans can either directly perceive or deduce from the GTS simulation.

The curvature is represented through an interpolation of the inverse radii of circles fitted through centerline points provided by GTS, as can be seen in Figure 1. They are equally distributed from 1.0 to 2.8 seconds into the future from the car's current position, estimated with the car's current speed.

D. Control Signal

We define the action vector $\boldsymbol{a}_t = [\delta_t, \omega_t]$ by two control signals: a steering angle $\delta_t \in [-\pi/6, \pi/6](rad)$ corresponding to the angle of the car's front wheels and a combined throttle-brake signal $\omega_t \in [-1, 1]$, where $\omega > 0$ denotes throttle and $\omega \leq 0$ represents braking. The magnitude of the throttle or the break is proportional to the absolute value $\|\omega\|$. For example, $\omega = 1$ is 100% throttle, $\omega = -1$ is 100% brake, and $\omega = 0$ is no throttle and no break. Combining the two signals reduces the complexity of the task. Since human expert recordings show that the best human strategies do not involve the simultaneous use of throttle and brake, we expect not to significantly restrict the agent by combining the two signals. To limit the outputs of the policy network to the valid range, we apply a *tanh* activation to the output layer as proposed in [5].

We use the automatic gearshift provided by GTS, since the used API does currently not allow to manually change gears.

[1] github.com/openai/spinningup

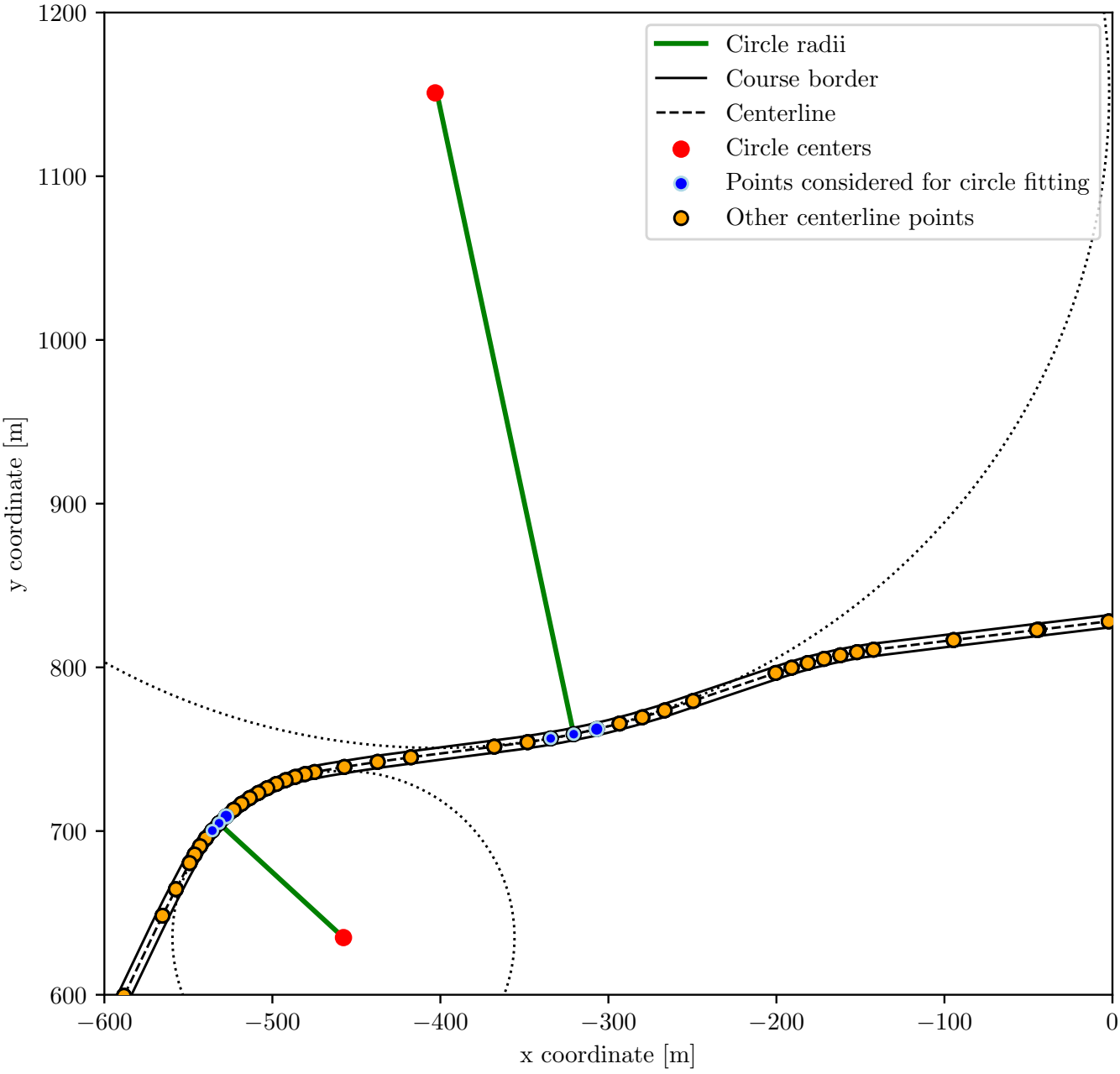

Fig. 1. **Construction of the curvature measurement.** To represent the curvature of the centerline, we make use of centerline points provided by the GTS simulation. The points are distributed such that regions with higher curvature are represented by more points. This allows getting a precise curvature measurement from looking at only three neighboring points. We make use of that property by defining the curvature for each centerline point by the inverse radius of the circle given by that point and its two neighbors. Right curvature is represented by negative inverted radii, left curvature by positive inverted radii. We then interpolate between centerline points to end up with a continous representation of the curvature over the course progress.

This option is also available to human players, but experienced players mostly use manual gearshift to have more control of when to shift.

E. Analysis of the Results by a Domain Expert

To improve our understanding of the achieved results, we invited *Gran Turismo* domain expert TG (name omitted for reasons of anonymity), who has achieved top performance in several national and international competitions, to race in our reference settings and compare his performance against our approach. TG competed in two of our three reference settings and achieved lap times in the top 0.36 and 0.23 percentile of our human reference data.

When asked for his opinion on the policies' driving style, TG stated:

> *"The policy drives very aggressively, but I think this is only possible through its precise actions. I could technically also drive the same trajectory, but in 999 out of a 1000 cases, me trying that trajectory results in wall contact which destroys my whole lap time*

and I would have to start a new lap from scratch."

F. Improvements over the built-in AI

In the 3 introduced reference settings the currently built-in AI is outperformed by a majority of human players as can be seen in Table I of the main manuscript. The built-in AI follows a pre-defined trajectory using a rule-based tracking approach, similar to other trajectory following approaches that have been widely studied in the control community [6], [7], [8]. Due to the non-linearity of the GTS dynamics, a slight deviation from such a trajectory strongly changes the new optimal trajectory, which makes it practically impossible to pre-compute and then track an optimal trajectory. The built-in AI solves this problem by using a cautious reference trajectory which includes a margin to the track's borders and allows recovery when deviating from the trajectory, as can be seen in Figure 8 of the main manuscript. This helps reducing the risk of contacting the track's side walls when deviating from the trajectory. However, it also leads to the trajectory's curves having smaller radii, forcing the AI to decelerate to not lose traction, resulting in curve exit speeds up to 50 km/h slower than those of our approach and the fastest human. The bottom of Figure 8 of the main manuscript shows how the built-in AI brakes in segments where neither our approach nor the fastest human brake, to be able to follow its reference trajectory. By learning a flexible driving policy without any explicit restrictions, our approach was able to overcome the shortcomings of the previous built-in AI and learn a driving policy more similar to that of a human expert driver regarding speed and path.

REFERENCES

[1] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[2] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[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] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributional policy gradients," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=SyZipzbCb

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, 2018, pp. 1856–1865.

[6] T. Hellstrom and O. Ringdahl, "Follow the past: a path-tracking algorithm for autonomous vehicles," *International journal of vehicle autonomous systems*, vol. 4, no. 2-4, pp. 216–224, 2006.

[7] P. Ritzer, C. Winter, and J. Brembeck, "Advanced path following control of an overactuated robotic vehicle," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1120–1125.

[8] J. Ni, J. Hu, and C. Xiang, "Robust path following control at driving/handling limits of an autonomous electric racecar," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5518–5526, 2019.