Model-based versus Model-free Deep Reinforcement Learning for Autonomous Racing Cars

Axel Brunnbauer[1*], Luigi Berducci[1*], Andreas Brandstätter[1*], Mathias Lechner[2], Ramin Hasani[3a], Daniela Rus[3], Radu Grosu[1]

***Abstract*— Despite the rich theoretical foundation of model-based deep reinforcement learning (RL) agents, their effectiveness in real-world robotics-applications is less studied and understood. In this paper we therefore investigate how such agents generalize to real-world autonomous-vehicle control-tasks, where advanced model-free deep RL algorithms fail. In particular, we set up a series of time-lap tasks for an F1TENTH racing robot, equipped with high-dimensional LiDAR sensors, on a set of test tracks with a gradual increase in their complexity. In this continuous-control setting, we show that model-based agents capable of learning in imagination, substantially outperform model-free agents with respect to performance, sample efficiency, successful task completion, and generalization. Moreover, we show that the generalization ability of model-based agents strongly depends on the observation-model choice. Finally, we provide extensive empirical evidence for the effectiveness of model-based agents provided with long enough memory horizons in sim2real tasks.**

I. Introduction

Deploying deep reinforcement-learning (RL) agents in the real-world is difficult. This is because they require running a significantly large amount of episodes to obtain reasonable performance [1]. This performance is only tractable in simulation environments. Subsequently, the agents should also overcome the challenges of transferring learned dynamics from simulation to the real world.

In RL settings, it was shown that if one learns a representation of the state-space model from high-dimensional input observations and use it as a predictive model to train policies in imagination, one gains performance, sample efficiency, and robustness [2]. This world-model algorithm [3] was called Dreamer, and it demonstrated performance in learning long-horizon visual control tasks by imagination.

Providing a model of the world (state transition probability) to the agents (even though in simulation) improves sample efficiency and, in some cases, the performance of a deep RL agent [3], [4], [2]. However, there are still open research questions to be investigated: For instance, how would world-model compartments improve the sim2real performance of RL agents in real-world applications? Where is the boundary between the generalization capability of model-based algorithms such as Dreamer, compared to advanced model-free agents such as soft actor-critic [5], distributional models [6], and policy gradient [7], [8], in sim2real applications?

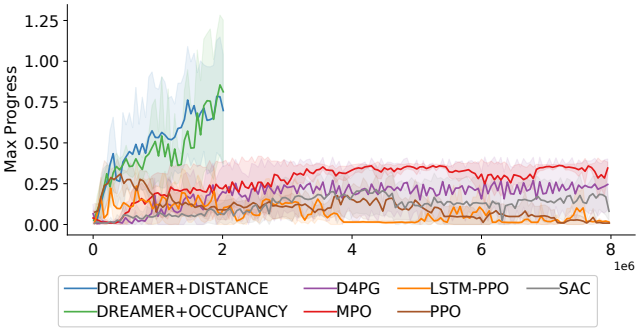

Fig. 1. Model-based deep-RL (Dreamer) solves autonomous racing in complex tracks where all advanced model-free methods fail.

In this paper, we aim to provide answers to the above questions. We construct a series of time-lap autonomous racing tasks with varying degrees of complexity for an F1TENTH robot. The task is to learn to drive autonomously directly from high-dimensional LiDAR inputs to finish a lap without collisions successfully.

We observe that as the complexity of the map increases, model-free agents get stuck in similar local minima and cannot learn to complete the task. Contrarily, Dreamer can learn a proper state transition model, and given enough imagination horizon, solves time-lap tasks of arbitrary complexity (See Fig. 1). Moreover, we discover that model-based techniques demonstrate desirable transferability: Dreamer agents trained on a single map can generalize well to test tracks they have never seen before. In summary, our **main contributions** are as follows:

- We demonstrate the effectiveness of advanced model-based deep RL compared to model-free agents in the real-world application of autonomous racing.
- We show the transferability of advanced model-based deep RL agents to the real-world applications where model-free agents fail.
- We empirically show that the learning performance and generalization ability of Dreamer in sim2real applications highly depends on the choice of the observation model, its resulting state transition probability model, and the imagination horizon.

II. Problem Definition

We set out to design deep RL models that are able to learn to autonomously complete time-lap racing tasks. Considering

* Indicates authors with equal contributions
[1] CPS, Technische Universität Wien (TU Wien), Austria
[2] Institute of Science and Technology Austria (IST Austria)
[3] CSAIL, Massachusetts Institute of Technology (MIT), MA, USA
[a] Correspondence to Ramin Hasani: rhasani@mit.edu

the real-world conditions, we formalize the problem as a Partially-Observable Markov Decision Process (POMDP).

Notation and terminology. A POMDP is defined as a tuple $(S, A, \Omega, O, \mathcal{T}, \mathcal{R})$, where S, A, Ω are the sets of states, actions and observations, respectively. O and $\mathcal{T}$ denote stochastic observation and transition functions, and $\mathcal{R}$ is a deterministic reward function. The transition function $\mathcal{T}$ models the system dynamics and includes its uncertainty. It is defined as a stochastic function $\mathcal{T} : S \times A \times S \to [0, 1]$ which returns the transition probability between two states by applying actions. The observation function O models the system's perception, including its uncertainty. Its stochastic formulation $O : S \times \Omega \to [0, 1]$ returns the probability of perceiving an observation in a given state. The reward function is deterministic $\mathcal{R} : S \times A \times S \to \mathbb{R}$, which returns the credit assigned to a transition. We discuss our reward shaping subsequently.

Racing-agent setup. In autonomous racing, the car drives along a race track by controlling its actuators with continuous actions $\boldsymbol{a} = (F, \alpha)^T$, where F is the motor force applied to reach a fixed target velocity, and α is the steering angle. In this scenario, the observations are range measurements obtained by a LiDAR sensor with 1080 range measurements evenly distributed over a 270° field of view (see Fig. 2).

How do we aim to solve this racing objective? Traditional control techniques model the state space as a set of continuous variables such as pose, velocity, and acceleration of the car. These quantities are usually estimated from observations by dedicated perception and filtering modules. State-space planning commonly uses sophisticated dynamics models for cars. This demands non-trivial system identification techniques. Contrarily, we aim to replace the entirety of such modules by learning an abstract state representation and transition model in a self-supervised manner by building upon recent advances of model-based RL [2].

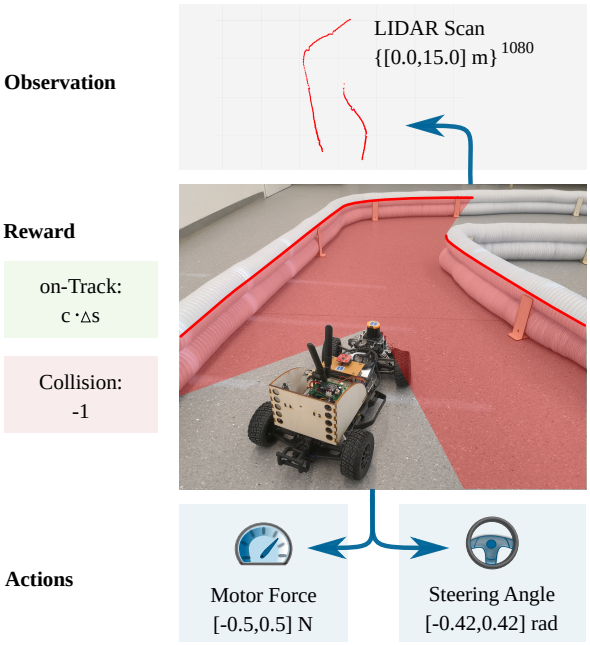

Fig. 2. Racing-agent setup: observations, actions and reward.

III. Related Work

We include works that are closely related to our contributions.

Model-free RL in robot control. Robot control has been a playground for RL algorithm for decades [1], [9]. Model-free algorithms [7], [5], [6] achieved state-of-the-art results in various continuous control tasks, but their inherent sample inefficiency [10] makes it challenging to apply them in real-world settings. Off-policy [11], [5] methods reduce the sample complexity by reusing old experience to some degree. However, they might induce errors when deployed in closed-loop with the real-world environment. Thus, model-free approaches are often trained in simulation before being deployed on real robots. This leads to difficulties when reproducing learned behaviors in the real setting [12], [13]. To overcome simulation mismatches, some approaches rely on domain randomization and subsequent fine-tuning. In order to learn more robust policies [14], other works refined the physics simulator [15], [16] or directly trained the policy on real-world robots [17], [18].

Model-based RL in robot control. Model-based approaches, on the other hand, can leverage their learned dynamics model by either planning or generating new training experience [19]. Problems arise when no sufficiently accurate dynamics model can be learned from data. An early representative of world-model algorithms is PILCO [20] which learned a Gaussian process from high-dimensional states. Recent works on world-models [3], [4], [2], [21] proved the feasibility of learning a dynamics model for POMDP's by using noisy observations instead of accurate states. They achieve better than state-of-the-art results on various simulated control tasks while being significantly more sample efficient. However, to the best of our knowledge, their application and challenges in real-world testbeds have been less attended, which we will explore in the present work.

Optimal control for autonomous racing. Autonomous racing has been an active field of research in the control community. Traditional control does not solve the problem in an end-to-end fashion but divides it into independent sub-problems: perception, planning, and control. Recent success in the context of Formula Student [22], [23], [24] has been achieved by Model Predictive Control (MPC) and by engineering the perception pipeline using sensor fusion of LiDAR and RGB cameras. Several approaches plan an optimal trajectory [25], [26], [27], [28]. However, these techniques usually require detailed global information (e.g., map) and an accurate dynamics model, not available in a POMDP setting, as in our case. Here, we use a fully automated process that learns an approximated dynamics model in an unsupervised fashion by only having access to LiDAR observations.

RL for autonomous racing. A large body of research made use of camera images and adopt model-free methods [29], [30], [31]. In [32], the authors used SAC to control a racing car in simulation by feeding the controller with state information and ad-hoc features about the road curvature. To deal with the

complexity of continuous action spaces, a common technique is to discretize the domain, but this approach is not scalable. Another recurrent trend consists of combining MPC and deep RL [33], [34]. However, MPC requires to efficiently perform sampling through the model to scale with respect to the time constraints. Conversely, the adoption of a policy network is more efficient and suitable for an online setting.

What about Imitation Learning? – Imitation learning consist of the process of learning observation to action mapping from supervised data [35]. Imitation learning allows for behavior cloning by data aggregation (Dagger) [36] via supervised learning modes or by inverse RL [37]. Imitation learning has been further adapted to imperfect [38] and incomplete demonstrations [39].

In the context of end-to-end control of autonomous vehicles, learning from expert demonstrations is the dominant choice of algorithm [40], [41], because other RL methods would require efficient simulation platforms to achieve desirable performance [42]. For autonomous racing, as we already have a sample efficient simulator available, we settle to use model-based RL algorithms over imitation learning.

IV. Formulating Dreamer for Autonomous Racing

In this section, we discuss the main considerations we had to address in order to design a dreamer architecture [2] for autonomous racing. For this purpose, we start by a brief recap of the Dreamer algorithm. We then present two observation models that we utilized to learn the latent-dynamics model. Finally, we formulate our specialized reward function.

Introduction. Dreamer is a model-based deep-RL algorithm that has recently shown very impressive performance in simulated standard RL benchmarks [2].

Dreamer learns an accurate model of the dynamics from high-dimensional observations in an unsupervised fashion and uses it to produce latent-state sequences to train an agent using an actor-critic algorithm [2]. Dreamer implements the dynamics model as a recurrent state-space model (RSSM) with both stochastic and deterministic components [4].

The dynamics-model consists of the four distributions below, where p highlights the interaction with the environment, q the use in latent imagination, too. All are implemented by deep neural networks [2]:

$$\text{Representation model:} \quad p_\theta(s_t|s_{t-1}, a_{t-1}, o_t) \tag{1}$$

$$\text{Observation model:} \quad q_\theta(o_t|s_t) \tag{2}$$

$$\text{Reward model:} \quad q_\theta(r_t|s_t) \tag{3}$$

$$\text{Transition model:} \quad q_\theta(s_t|s_{t-1}, a_{t-1}) \tag{4}$$

All components are jointly optimized to maximize the variational lower-bound as follows, where $D_{\mathrm{KL}}(P||Q)$ is the Kullback–Leibler divergence of distributions P and Q:

$$\mathcal{J}_O^t = \ln q(o_t|s_t)$$
$$\mathcal{J}_R^t = \ln q(r_t|s_t)$$
$$\mathcal{J}_D^t = -\beta D_{\mathrm{KL}}(p(s_t|s_{t-1}, a_{t-1}, o_t)||q(s_t|s_{t-1}, a_{t-1}))$$
$$\mathcal{J}_{REC} = \mathbb{E}_p\left[\sum_t \mathcal{J}_O^t + \mathcal{J}_R^t + \mathcal{J}_D^t\right]$$

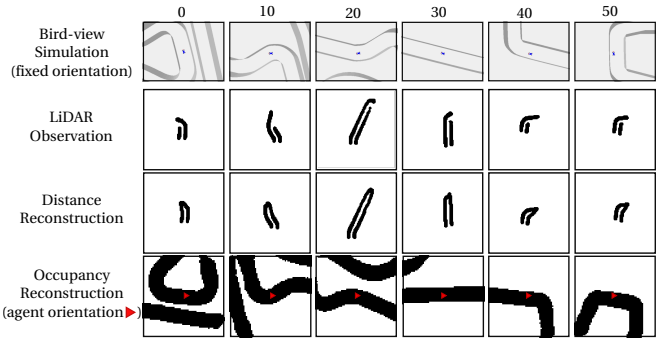

Fig. 3. Observation models and their reconstruction methods. Observations are in agent coordinates. **Row 1:** bird-view of the racecar in simulation, **row 2:** LiDAR scan in racecar coordinates, **row 3:** reconstructed LiDAR scan, **row 4:** reconstructed local occupancy map

In contrast to [2], our representation model uses a multilayer perceptron (MLP) to encode the observations, instead of convolutional layers. The reason is that LiDAR observations have 1080 dimensions, and further compression does not show better performance in our empirical evaluation.

Observation model. This model determines how the observations are represented by the RL algorithm. It is a Dreamer component that facilitates learning a good latent representation. The model is used only in the training phase to provide learning signals to the representation model. In this work, we first propose two observation models: one based on reconstructing distance measurements, and the other is based on reconstructing occupancy maps, respectively. We then highlight their importance in the generalization performance of a Dreamer agent.

Distance reconstruction is formulated in terms of the distance measured by each LiDAR ray at every time step. We call the resulting agent Dreamer-Distance. In this case, the observation is both the input of the representation-model and the output of the observation model. The observation model is an MLP that computes the mean and standard deviation of a Gaussian distribution for each ray.

Occupancy reconstruction builds the surrounding occupancy map at each time step. We call the resulting agent Dreamer-Occupancy. Specifically, the reconstructed area is centered on the agent position and has a radius of 5 meters. The observation-model is a convolutional network that computes the mean of a Bernoulli distribution. Each value is an indicator of the pixel occupancy in the surrounding map. In Figure 3 we show a snapshot of the simulation, the associated LiDAR observation, and the two reconstructions.

Note that the input of the representation model is slightly different from the output of both observation-models.

Actor-critic on latent imagination. Having first learned a world model by using complex observations, Dreamer then learns a policy purely on imagined sequences [2]. This approach allows Dreamer to efficiently produce thousands of training sequences, without requiring any direct interaction with the environment. This results in safe and data-efficient learning. Starting from these imagined sequences, Dreamer

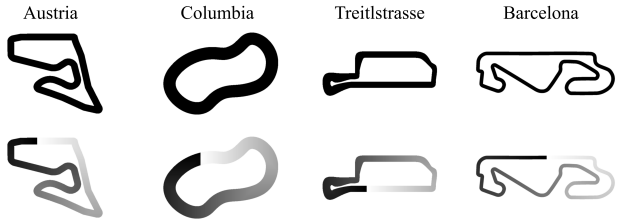

Fig. 4. Top-view of the race tracks and relative Progress Maps adopted for reward design. Austria (length = 79.45m), Columbia (61.2m), Treitlstrasse (51.65m) are used for training, Barcelona (201m) is used only for evaluation.

uses an actor-critic algorithm to train the agent. The action model (policy) aims to synthesize the optimal action in each latent state, and the value model (critic) estimates the value of each latent state [2]:

$$\text{Action model:} \qquad q_\phi(a_t|s_t) \tag{5}$$

$$\text{Value model:} \qquad q_\psi(v_t|s_t) \tag{6}$$

Among the various techniques for approximating state values and balancing bias and variance [1], Dreamer [2] uses an exponentially weighted sum over different horizons. Moreover, the training process leverages the availability of a learned dynamics model by back-propagating through it. This leads to more informative gradients.

Reward shaping. The reward design (or shaping) is critical in determining the quality of the obtained policy. The sparsity of rewards over long episodes makes the learning problem challenging. In Figure 4 we show precomputed gridmaps for each of our tracks. These gridmaps show the normalized distance, which we interpret as progress, from the starting line in race direction.

The reward signal we propose is computed by

$$c * |s_t - s_{t-1}| = c * \Delta s_t. \tag{7}$$

In Equation 7, s_t denotes the progress value at time t and c is a constant scalar. In case of collision, the agent receives a penalty, and the episode terminates (see also Figure 2).

V. Experiments

Here, we describe our experimental setup and evaluate the performance of model-free versus model-based methods.

A. Experimental Setup

We train the models in simulation and transfer them to real cars. Here, we provide an overview of the simulation and real-world setup as well as the training process.

Simulation environment. To simulate a car model, we use the open-source physics engine PyBullet [43]. The car model is a rigid body system based on the URDF model in [44]. Our training environment can simulate a broad set of sensory inputs, such as LiDAR sensors, RGB cameras, and odometry. The model is actuated by applying force to the steering and acceleration joints. See supplements for more details.

Training pipeline. During training in simulation, we place the agent randomly on the track at the beginning of an episode. Each training episode has a maximum length of 2000 timesteps, resulting in 20 seconds of real-time experience. Agents learn to directly maximize the progress covered in a small, predefined time interval. Observations and actions are normalized. To account for latency experienced during testing and to increase the effectiveness of actions, we repeat each action multiple times. We regularly evaluate the agent by placing it on a fixed starting position for each track and let it run for at most 4000 timesteps (i.e., 40 seconds) and average the maximum progress reached over five consecutive trials. Dreamer is trained for 2 million timesteps. The model-free baselines are trained for 8 million timesteps.

TABLE I
Track characteristics.

Track	Min. Width	Length	Min. Radius
Austria	1.86 m	79.45 m	2.78 m
Columbia	3.53 m	61.20 m	7.68 m
Treitlstrasse	0.89 m	51.65 m	3.55 m
Barcelona	1.86 m	201.00 m	2.98 m

Hardware setup. The hardware platform for the race car is based on the F1TENTH race series [45]. It consists of an off-the-shelf model race car chassis, with a Traxxas Velineon 3351R brushless DC electric motor, which is driven by a VESC 6 MkIV electronic speed controller (ESC). Computation and control tasks are performed on an NVIDIA Jetson TX2. The environment is sensed by a Hokuyo UST-10LX LiDAR sensor. It is running Ubuntu 18.04 as the base operating system. Robot Operating System (ROS) [46] is used to provide an infrastructure for sending and receiving messages and for the hardware abstraction. Our dreamer agent is running inside a Docker container. A ROS interface node is used to translate observation and action messages. The motor force commands are processed by integration in order to get the desired speed values. For the steering commands, we added an adaptive low pass filter in order to protect the servo from high frequent steering operations.

Track difficulty. In order to compare the difficulty of the tracks, we classified them according to several characteristics (as also discussed in [47] and [48]). In particular, we measure the minimal track width, the track length (shortest path), and its minimum curve radius. For curves, we measure the radius of the largest circle that tangentially touches the outside of the curve and also touches the inside of the curve. In Table I we summarize the characteristics of the the tracks.

B. Experimental Evaluation

In this section, we first introduce the baselines and then discuss in detail the performance of the algorithms in various racing scenarios.

Model-free baselines. We compare the performance of the Dreamer agent against the following advanced model-free baselines: D4PG, an enhanced version of DDPG [6], MPO, a stable off-policy algorithm [8], SAC, an off-policy actor-critic algorithm with less sensitivity to hyperparameters [17], PPO,

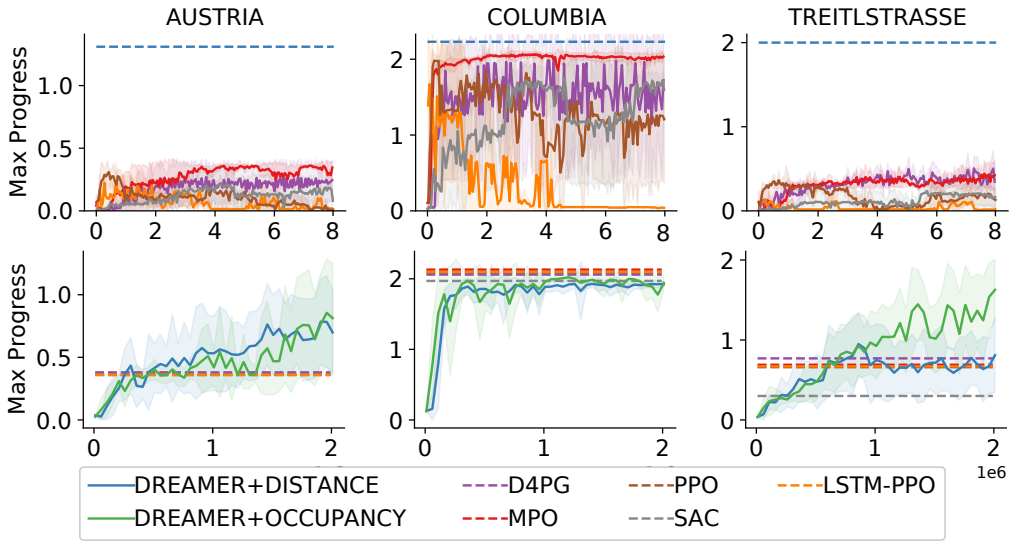

Fig. 5. Learning curves of model-free methods (*top row*) over $8M$ steps and Dreamers (*bottom row*) over $2M$ steps. The dashed lines report the maximum performance obtained by the other algorithms as baselines. n=5

an on-policy algorithm [7], and PPO-LSTM, the recurrent version of PPO using long short-term memory [49]. PPO-LSTM is chosen as a baseline since policies built by recurrent neural network approximators demonstrate remarkable performance in learning to control [50], [51], [52], [53].

We conduct three different experiments: I) Evaluation of the learning curves of Dreamer and model-free algorithms in simulation, II) Evaluation of the generalization ability by testing the trained models in simulation, and III) Evaluation of the transferability on our real testing platform.

Learning performance and sample efficiency. In Figure 5 we give the learning curves of the model-free and model-based algorithms in 3 different tracks. Except for the simple track Columbia, all advanced model-free methods fail to successfully perform one lap in more complex maps. On the other hand, the Dreamer agents efficiently learn to complete the tasks regardless of the degrees of complexity of the tracks.

As shown in Figure 5, the performance of the Dreamer agents equipped with distance and occupancy reconstructions are comparable in Austria and Columbia. However, the experiments on Treitlstrasse show better performance with occupancy-map reconstruction, with which the agent can almost complete two full laps over the evaluation-time window. This experiment suggests that the occupancy-map reconstruction speeds up the training process. However, it biases the learning process on the track on which it was trained. This affects the policy's generalizability, as illustrated in the next experiment.

What length of imagination horizon? Figure 6 shows that as we increase the imagination horizon for a Dreamer agent, it learns to perform better. Our experiments show that the horizon plays an important role in learning long-range tasks.

Performance on unseen tracks. In this experiment, we aim to evaluate the cross-track generalizability of the learned policies and demonstrate the domain adaptation skills of Dreamer. We trained polices on complex tracks, Austria (AUT) and Treitlstrasse (TRT) individually, and reported their test performance on other unseen tracks: Columbia (COL) and Barcelona (BRC). We compared the performance of the best performing model-free policy, MPO, to Dreamer-Distance and Dreamer-Occupancy policies.

As shown in Figure 7 (top-left and top-right), while all algorithms can complete the simple track, COL, only Dreamer can generalize the racing task and transfer the learned skills to other complex tracks. Generally, policies trained in AUT achieve better performance as they can complete at least one lap in each other track.

Conversely, even if TRT contains very challenging turns, the trained agents show bad performance on unseen complex tracks (AUT). The reason might be that most turns in TRT have the same direction, and consequently, the trained policy cannot generalize to altered scenarios.

Moreover, we observed that the Dreamer-Occupancy agent shows lower generalization skills compared to Dreamer-Distance. The learned policy results in a more aggressive strategy, presenting a lower lap-time as shown in Figure 7 (bottom-left and bottom-right). However, the car often gets dangerously close to the track walls. This results in unsafe behavior that

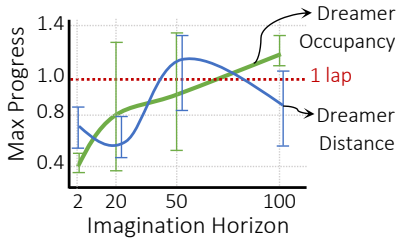

Fig. 6. Dreamer's performance using different reconstructions for different values of horizon. The batch length is fixed to50 and action repeat is 4. The figure reports the mean performance over 5 trials.

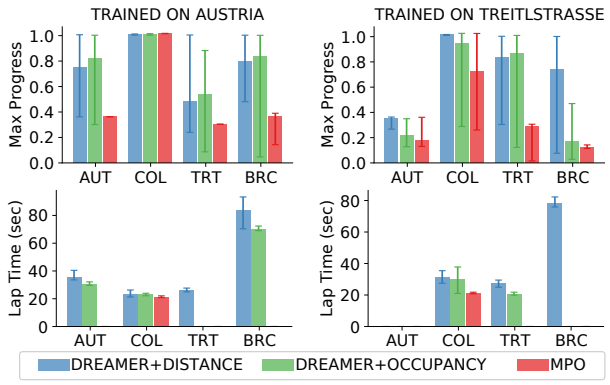

Fig. 7. Maximum progress and lap time of trained models over different tracks in simulation. The bars show the result averaged over 10 episodes on each track. The delimiters show the minimum and maximum achieved. For Lap-Time results, we consider only the run completing one full lap.

increases the probability of collisions.

In conclusion, these observations suggest that reconstructing the occupancy area would bias the agent towards learning the track on which it was trained. This results in faster learning but induces a less robust behavior to changes in the domain and worsens generalization.

Sim2Real transfer. In this experiment, we evaluate Dreamer policies with respect to their sim2real transferability. We test trained dreamers deployed on the car in a physical test-track. A video[1] demonstrating the driving performance of the Dreamer agent is supplied with the submission. It presents the Dreamer agent completing a full lap in TRT in the forward direction. Then, to evaluate its generalization, we tested the agent in reversed direction. We observe that even if the agent was not trained in the reversed direction and not directly on the TRT track, the agent can complete a successful lap. Finally, we placed two obstacles after the most challenging turn and ran the agent in this configuration. The Dreamer agents can complete the lap and securely avoid obstacles.

VI. Conclusions

We showed that Dreamer, a model-based deep RL algorithm, performs significantly better in driving autonomous race cars in complex tracks compared to advanced model-free agents. We empirically show that increasing an agent's imagination horizon accounts for this discrepancy. We showed how observation models affect generalization and domain adaptation of learned policies, and that Dreamer agents can enable robust autonomy in real-world settings.[2] [3]

Acknowledgments

L.B. was supported by the Doctoral College Resilient Embedded Systems. M.L. was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award). R.H. and D.R. are supported by Boeing. R.G. was partially supported by the Horizon-2020 ECSEL Project grant No. 783163 (iDev40) and A.B. by FFG Project ADEX.

[1]The video can be viewed at: `https://youtu.be/IlN3vJxC30w` doi: 10.5281/zenodo.4577412

[2]All code, data and appendix are available at: `https://github.com/CPS-TUWien/racing_dreamer`

[3]All Experimental logs are archived at: doi: 10.5281/zenodo.4588599

References

[1] R. S. Sutton and A. G. Barto, *RL: An introduction*. MIT press, 2018.

[2] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv:1912.01603*, 2019.

[3] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf

[4] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *Int. Conf. on Machine Learning*. PMLR, 2019, pp. 2555–2565.

[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 *Int. Conf. on Machine Learning*, 2018, pp. 1861–1870.

[6] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributed Distributional Deterministic Policy Gradients," in *International Conference on Learning Representations*, 2018.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[8] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," in *International Conference on Learning Representations* , 2018. [Online]. Available: https://openreview.net/forum?id=S1ANxQW0b

[9] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4082–4093.

[10] Y. Yu, "Towards sample efficient reinforcement learning," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence* , ser. IJCAI'18. AAAI Press, 2018, p. 5739–5743.

[11] 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.

[12] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, "The ingredients of real world robotic reinforcement learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rJe2syrtvS

[13] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep RL for Robotics: a Survey," in*IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2018.8460528

[15] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," in *Proceedings of Robotics: Science and Systems* , Pittsburgh, Pennsylvania, June 2018.

[16] M. Kaspar, J. D. Muñoz Osorio, and J. Bock, "Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4383–4388.

[17] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[18] A. Singh, L. Yang, C. Finn, and S. Levine, "End-To-End Robotic Reinforcement Learning without Reward Engineering," in *Proceedings of Robotics: Science and Systems*, 2019.

[19] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bull.*, vol. 2, no. 4, p. 160–163, Jul. 1991. [Online]. Available: https://doi.org/10.1145/122344.122377

[20] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *Proceedings of the 28th International Conference on Machine Learning* ser. ICML'11. Madison, WI, USA: Omnipress, 2011, p. 465–472.

[21] W. Schwarting, T. Seyde, I. Gilitschenski, L. Liebenwein, R. Sander, S. Karaman, and D. Rus, "Deep latent competition: Learning to race using visual control policies in latent space," *arXiv preprint arXiv:2102.09812*, 2021.

[22] J. Kabzan, M. de la Iglesia Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart, "Amz driverless: The full autonomous racing system," 2019.

[23] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363 – 3370, 2019-10.

[24] L. Andresen, A. Brandemuehl, A. Honger, B. Kuan, N. Vödisch, H. Blum, V. Reijgwart, L. Bernreiter, L. Schaupp, J. J. Chung, M. Burki, M. R. Oswald, R. Siegwart, and A. Gawel, "Accurate Mapping and Planning for Autonomous Racing," in *2020 IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4743–4749.

[25] E. Velenis and P. Tsiotras, "Minimum Time vs Maximum Exit Velocity Path Optimization During Cornering," in *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, vol. 1, 2005, pp. 355–360.

[26] A. Rucco, G. Notarstefano, and J. Hauser, "An Efficient Minimum-Time Trajectory Generation Strategy for Two-Track Car Vehicles," *IEEE Trans on Control Systems Tech*, vol. 23, no. 4, pp. 1505–1519, 2015.

[27] J. P. Timings and D. J. Cole, "Minimum maneuver time calculation using convex optimization," *Journal of Dynamic Systems, Measurement, and Control*, vol. 135, no. 3, 2013.

[28] J. L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 2397–2403.

[29] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2070–2075.

[30] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *2007 Frontiers in the Convergence of Bioscience and Information Technologies*, 2007, pp. 645–650.

[31] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8248–8254.

[32] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Duerr, "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning," *arXiv preprint arXiv:2008.07971*, 2020. [Online]. Available: https://arxiv.org/abs/2008.07971

[33] G. Bellegarda and K. Byl, "An Online Training Method for Augmenting MPC with Deep Reinforcement Learning," in *2020 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 5453–5459.

[34] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1714–1721.

[35] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[36] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[37] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[38] Y.-H. Wu, N. Charoenphakdee, H. Bao, V. Tangkaratt, and M. Sugiyama, "Imitation learning from imperfect demonstration," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6818–6827.

[39] M. Sun and X. Ma, "Adversarial imitation learning from incomplete demonstrations," *arXiv preprint arXiv:1905.12310*, 2019.

[40] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, no. 10, pp. 642–652, 2020.

[41] M. Lechner, R. Hasani, M. Zimmer, T. A. Henzinger, and R. Grosu, "Designing worm-inspired neural networks for interpretable robotic control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 87–94.

[42] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020.

[43] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[44] V. S. Babu and M. Behl, "f1tenth. dev-an open-source ros based f1/10 autonomous racing simulator," in *16th Int. Conf. on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 1614–1620.

[45] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, "Tunercar: A superoptimization toolchain for autonomous racing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5356–5362.

[46] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[47] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, "Race driver model," *Computers & Structures*, vol. 86, no. 13, pp. 1503–1516, 2008, structural Optimization. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045794908000163

[48] F. Lamiraux and J. . Lammond, "Smooth motion planning for car-like vehicles," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 498–501, 2001.

[49] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[50] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "Liquid time-constant networks," *arXiv preprint arXiv:2006.04439*, 2020.

[51] M. Lechner, R. Hasani, D. Rus, and R. Grosu, "Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5446–5452.

[52] R. Hasani, A. Amini, M. Lechner, F. Naser, R. Grosu, and D. Rus, "Response characterization for auditing cell dynamics in long short-term memory networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[53] M. Lechner and R. Hasani, "Learning long-term dependencies in irregularly-sampled time series," *arXiv preprint arXiv:2006.04418*, 2020.

[54] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A research framework for distributed reinforcement learning," 2020.

[55] 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.

[56] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.

[57] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[58] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning* ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: http://proceedings.mlr.press/v28/bergstra13.html

[59] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[60] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm: "follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889012000838

[61] U. of North Carolina, "UNC CS team wins F1/10 autonomous racing challenge," https://cs.unc.edu/news/unc-cs-team-wins-f1tenth-autonomous-racing-challenge/, 2019.

Appendix

S1. Hyperparameters

A. Dreamer

For generating initial training data, we use a Follow-the-Gap approach (see section S2 for details). This choice is motivated by the fact that random actions will not lead to actual driving behavior and that data of longer trajectories contain more useful information to train the dynamics model.

In our experiments, we focused on the sensitivity of Dreamer to variations of the parameters *Imagination Horizon* and *Action Repeat*. We carried out our experiments by fixing one configuration and systematically evaluate variations of the other parameters. The configuration details are described in Table S1, where the column *Value* reports the configuration that we used to produce the learning curves presented in the main text.

Parameter	Value	Search Space
Initial Dataset Size	5000 steps	-
Update Steps	100	-
Batch Size	50	-
Sequence Length	50	-
Action Repeat	8	4, 8
Imagination Horizon	15	2, 20, 50, 100
Learning Rate θ	6e-4	-
Learning Rate ψ	8e-5	-
Learning Rate ϕ	8e-5	-
Discount	0.99	-
λ	0.95	-

TABLE S1

Hyperparameters for Dreamer

B. Model Free Baselines

To compare the performance of Dreamer to other RL algorithms, we use stable implementations of model-free algorithms provided by multiple open-source RL frameworks. Specifically, we used MPO, D4PG from ACME[54] and SAC, PPO, PPO-LSTM from Stable Baselines [55], [56].

For the model-free baseline algorithms MPO, D4PG, PPO and SAC we choose the same neural network architecture as for the latent actor-critic model of Dreamer. The recurrent version of PPO uses a shared recurrent observation network (see Table S3). For MPO and D4PG we choose ELU activation functions, for PPO(-LSTM) and SAC we use the default settings of the implementation [55], [56]. For all details, please refer to the code-repository: `https://github.com/CPS-TUWien/racing_dreamer`.

Module	Layers
Actor Network	MLP with 4 Layers, 400 neurons each
Critic Network	MLP with 3 Layers, 400 neurons each

TABLE S2

MPO, D4PG, PPO and SAC Architectures

Module	Layers
Shared Observation Network	MLP with 2 Layers, 200 neurons each LSTM, 256 Cells
Actor Network	MLP with 3 Layers, 400 neurons each
Critic Network	MLP with 2 Layers, 400 neurons each

TABLE S3

PPO-LSTM Architecture

Tables S4 to S7 show the sets of parameters for the algorithms as well as the search space of the hyperparameter study, which we conducted for MPO, D4PG and PPO. It turned out that in most of the cases, the default parameters worked best, as

they allowed a more stable learning. The study optimizes hyperparameters of the algorithms by doing 100 training runs for 1 million timesteps on the Austria track. The objective value is the average progress achieved within at most 4000 timesteps over 5 consecutive runs. As an optimization procedure we make use of the Tree-structured Parzen Estimator [57], [58] and HyperBand for pruning unpromising examples. We use the Optuna hyperparameter optimization library to conduct these experiments [59].

Parameter	Value	Search Space
Policy Learning Rate	10^{-4}	$[10^{-5}, 10^{-3}]$
Critic Learning Rate	10^{-4}	$[10^{-5}, 10^{-3}]$
Batch Size	256	-
Epsilon	0.1	-
Epsilon Mean	10^{-3}	$[5\times10^{-4}, 10^{-3}]$
Epsilon Stddev.	10^{-6}	$[5\times10^{-7}, 10^{-5}]$
Replay Buffer Size	5×10^{5}	-

TABLE S4
HYPERPARAMETERS FOR MPO [8]

Parameter	Value	Search Space
Policy Learning Rate	10^{-4}	$[10^{-5}, 10^{-3}]$
Critic Learning Rate	10^{-4}	$[10^{-5}, 10^{-3}]$
Batch Size	256	-
Sigma	0.3	-
Atoms	51	-
Discount	0.99	-
Replay Buffer Size	5×10^{5}	-

TABLE S5
HYPERPARAMETERS FOR D4PG [6]

Parameter	Value	Search Space
Learning Rate	3×10^{-4}	$[10^{-5}, 3\times10^{-3}]$
Batch Size	64	-
Discount	0.99	-
GAE-λ	0.95	-
Clip Range	0.2	$[0.1, 0.3]$
Entropy Coef.	0.0	$[0.0, 0.01]$
Value Function Coef.	0.5	$[0.5, 1.0]$

TABLE S6
HYPERPARAMETERS FOR PPO [7] AND PPO-LSTM

S2. COMPARISON WITH FOLLOW THE GAP

Follow the Gap. Achieving good driving performance without using a map is difficult because of the lack of global information on the track. In 2012, a novel method called Follow the Gap (FTG) was introduced [60].

The algorithm proposes a simple obstacle-avoidance strategy: given the sensed distances by the LiDAR, FTG chooses the steering angle to avoid the closest obstacle and trying to identify the largest gap in the sensed distances. Concerning the velocity, FTG adaptively tunes the reference velocity to be proportional to the maximum distance scanned in front of the car. The velocity ranges within a desired interval to avoid the car goes out of control.

This strategy is safe by construction, extraordinarily effective, and easy to implement on real cars. In fact, FTG was the base algorithm recently adopted by UNC to win the F1TENTH Montreal 2019 Competition [61]. For these reasons, we use FTG as a baseline for some experiments conducted in simulation.

Dreamer versus FTG. Figure S1 shows the performance of our Dreamer implementations, MPO, and FTG concerning maximum progress and lap-time averaged over 10 episodes.

As already explained in the main text, the training track and observation model play an important role in the training process. For this reason, some configurations cannot complete one full lap, resulting as missing bars in Figure S1 (bottom).

Parameter	Value
Learning Rate	3×10^{-4}
Batch Size	256
Tau	0.005
Discount	0.99
Replay Buffer Size	5×10^{5}

TABLE S7

HYPERPARAMETERS FOR SAC [5]

However, considering the Dreamer agents trained in Austria, a few remarks are worth mentioning. In fact, despite FTG is a programmed algorithm and its parameter has been tuned to achieve good performance and lap time comparable with our Dreamer agent, Figure S1 shows that FTG also struggles in completing Austria, Treitlstrasse, and Barcelona.
The maximum-progress achieved by Dreamer-Distance is comparable to FTG. The lap time of FTG is slightly better, and the overall performance shows less variance. This is not surprising considering that we optimize this algorithm for these tasks.

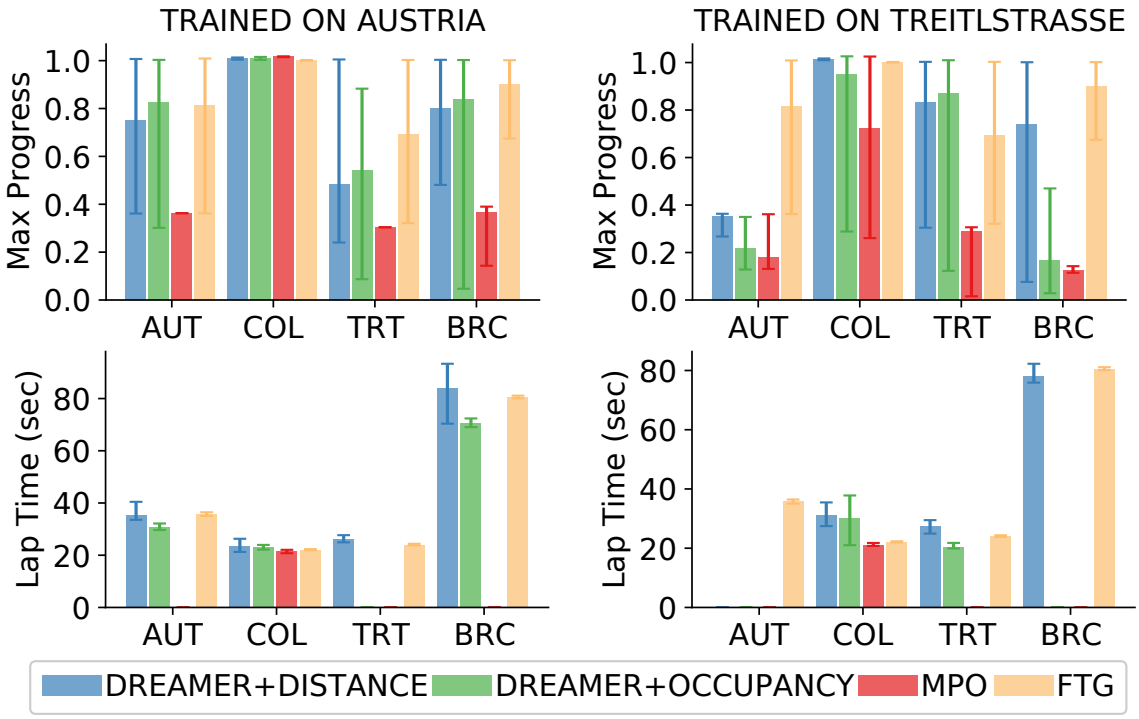

Fig. S1. Comparison of average performance and lap-time on unseen track over 10 evaluation episodes. The bar reports the average metric achieved over several race tracks and the minimum and maximum are described by the delimiters.

Analysis of trajectories. One of the drawbacks of FTG is its zig-zagging behavior resulting from its gap-oriented policy. To analyze the policy learned by our Dreamer agent, we collect 10 trajectories by running the best agent of each configuration on all the evaluation tracks. The visualization of the trajectories is reported in Figure S2 and we can identify the problematic turns that cause collisions in each tracks. Note that to avoid to overload the visualization, we select 10 trajectories, while in the previous experiments in Figure S1 we average the performance over multiple agents, obtaining much larger statistics.
The zig-zagging behavior of FTG is evident in Austria, Treitlstrasse, and Barcelona, where the car continually changes its orientation in the straight part of the track. Conversely in wider tracks like Columbia, FTG shows a smoother trajectory and drives close to the internal wall.
Dreamer's trajectories result more stable and centered in the track in the straight sections. Figure S2 also highlights that Dreamer reduces the velocity in the turns' proximity. The mitigation of these sharp changes in velocity could benefit the racing performance and will be addressed in future work.

S3. RACE CAR HARDWARE

The hardware platform for the race car is shown in Figure S3 (left). It is based on the F1TENTH race series [45]. It consists of a off-the-shelf model race car chassis *Traxxas Ford Fiesta ST*. Propulsion is provided by a brushless DC electric motor

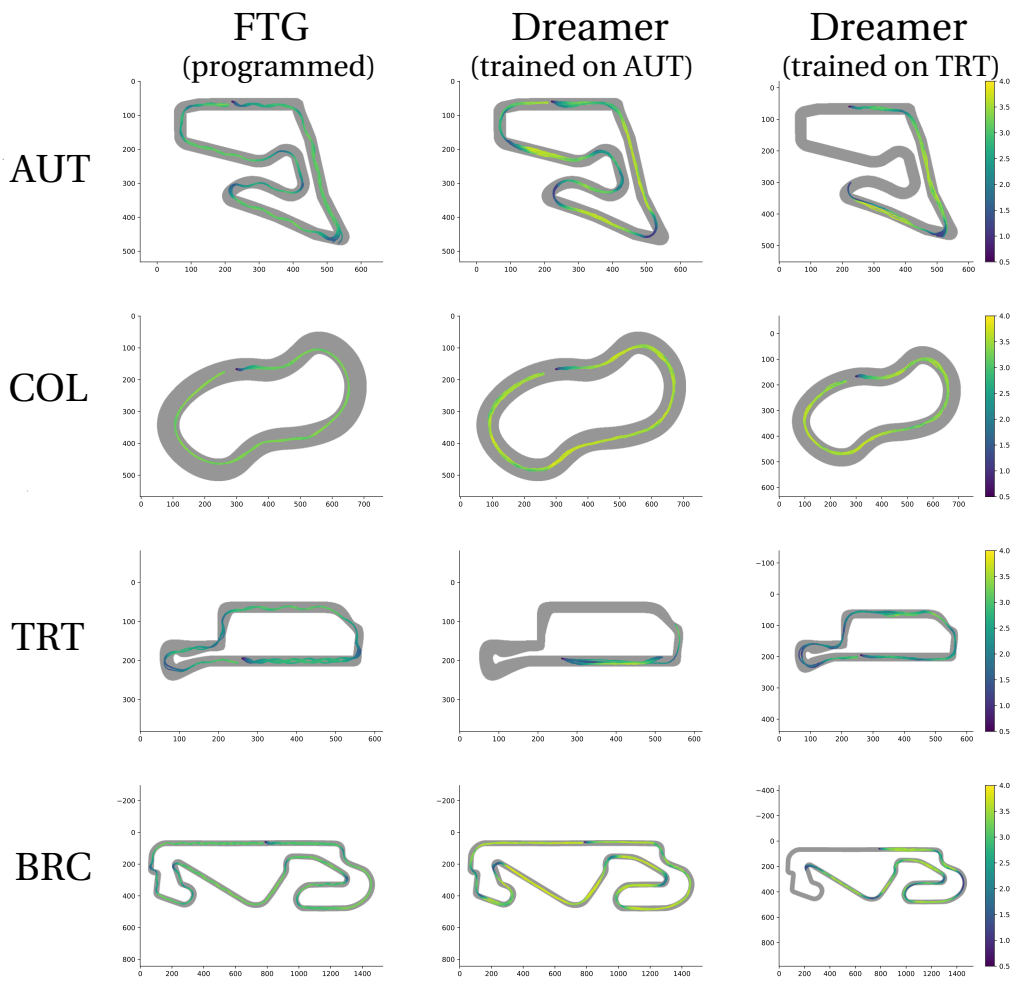

Fig. S2. Comparison of trajectories obtained by the trained policies on Austria (AUT) and Treitlstrasse (TRT) with the programmed method Follow The Gap (FTG). For each track and agent, we show 10 trajectories to highlight the section of the track, which results in difficulty for the agent. The color of the trajectories denotes the speed in m/s.

Traxxas Velineon 3351R which is driven by a *VESC 6 MkIV* electronic speed controller (ESC). Computation and control tasks are performed on a *NVIDIA Jetson TX2* embedded computing platform. Supported by a *Tech Orbitty Carrier* adapter board it interfaces the sensors and actuators by Ethernet, USB and RS-232. The environment is sensed by a *Hokuyo UST-10LX* LiDAR sensor. The *SparkFun 9DoF Razor IMU M0* inertial measurement unit provides data about the acceleration and angular velocity of the race car (these measurements were not used throughout this work). For human interaction during the operation of the car there is a *Logitech F710* Gamepad controller attached. This input is used to stop the car in case of emergency.

S4. Race Car Software Setup

Figure S3 (right) shows the software setup of the actual racing car. It is running Ubuntu 18.04 as the base operating system and uses Robot Operating System (ROS) [46] to provide an infrastructure for sending and receiving messages and for the hardware abstraction. Third-party device drivers (*urg_node*) for the LiDAR sensor and VESC (*vesc_node*) are used as an interface between the hardware components and ROS. Auxiliary nodes for specific tasks (e.g., emergency stop, translating messages) are based on the general software setup of [45]. In order to have a similar environment in simulation and on the racing car on an operating system level, we are using a docker container to run our dreamer agent. Inside this docker container A ROS interface node is used to translate observation and action messages. The motor force commands are processed by integration in order to get the desired speed values. For the steering commands, we added an adaptive low pass filter in order to protect the servo from high frequent steering operations.

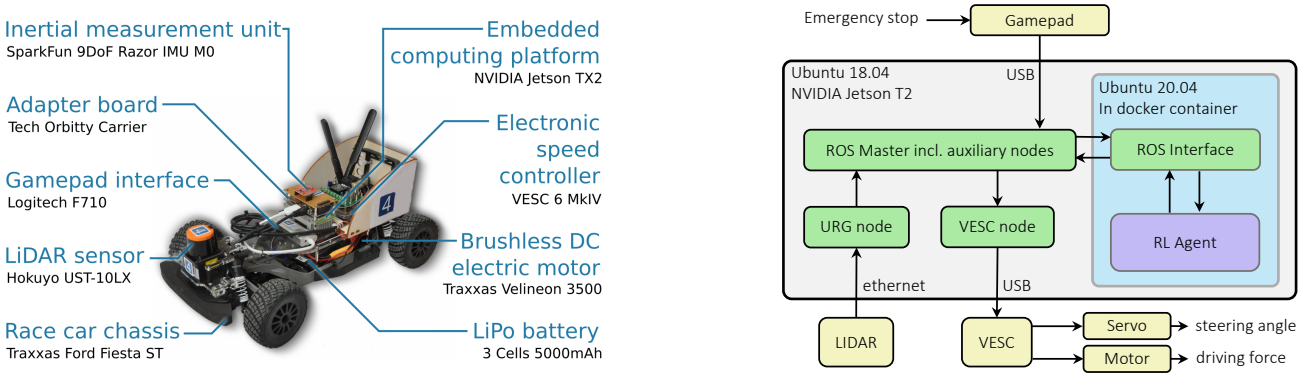

Fig. S3. Race car hardware platform and software architecture.