I. INTRODUCTION

AUTONOMOUS racing on resource-constrained hardware pushes the boundaries of algorithmic design and implementation in perception, planning, and control [1], [2], [3]. Thus, it is a valuable asset for researchers to push the limits of autonomous driving [4], [5], as it can lead to many benefits, such as enhancing road safety, reducing carbon emissions, transporting the mobility-impaired, and reducing driving-related stress [6], [7]. Due to the ambitious research challenges, in recent years, many autonomous racing competitions have emerged and received considerable attention in the fields of robotics and artificial intelligence (AI). For instance, the formula student driverless [1] and the AWS DeepRacer league [8] are popular and followed by tens of teams. Among other competitions, the FITENTH racing platform [9] is gaining popularity by organizing semiregular autonomous racing competitions involving a physical race car on a scale of 1:10. As the standardized platform offers little room for improvement on the hardware side, the main challenges are raised on the algorithmic side [10], where the resource-constrained processor with limited memory and computational resources makes the algorithmic design even more challenging. Namely, the embedded control layer becomes the key focus of development, as the system in itself is highly nonlinear, and the behavior of the car must be taken into consideration at the edge of stability [11], [12].
Current state-of-the-art (SotA) racing controllers utilize optimal control methods, such as model predictive control (MPC) [1], [4], [5], [11]. While MPC can guarantee the optimality of the planned trajectory and tracking within its receding horizon, it heavily relies on the accuracy of the modeling parameters, and as shown in [13], heuristic strategies can outperform MPC even if the latter contains more information about the controlled system. In particular, in the context of autonomous car racing, the model inaccuracies of the lateral tire forces are highly critical [14], [15]. These forces are notoriously difficult to model, and the tires’ behavior is highly nonlinear [16], [17]. In real racing scenarios, a tire modeling mismatch is very likely to occur, as high wear, tear, and weight changes modify the initial parameters [17]. While there exist several previous works that have attempted to address this issue using learning-based methods [18], [19], [20] for MPC, we assess and highlight the feasibility and performance of a reinforcement learning (RL) approach, that allows robust control behavior without the need of complex model-contextual optimization in MPC.

RL [21] methods offer a machine learning (ML)-based solution that showed to be able to handle complex robotic and control tasks, such as plasma control [22], hand manipulation [23], quadrupedal locomotion [24], and autonomous racing as given by Fuchs et al. [25] and Wurman et al. [26], where they apply RL to outperform professional human drivers in the setting of a highly realistic video game. Brunnbauer et al. [27], instead, show that model-based RL architectures prove to be better at generalizing to new driving tasks and look more promising when trying to overcome the Sim2Real gap. Chisari et al. [28] then apply a regularization strategy to the ML agent and show that this substantially improves the Sim2Real capabilities. The mentioned architectures are end-to-end learned, meaning that they learn the optimal control policy directly from sensory input. Recent previous works [29], [30] highlight that to be able to derive control policies from raw sensory data, relevant semantics (e.g., track features) must be automatically learned by the system, and propose learning-based enhancements to improve the learning process, e.g., as given by De Bruin et al. [29], where they show that using an autoencoder structure can improve the reward obtained on unseen tracks by three times. On the other hand, such features can be extracted with traditional control procedures to improve the robustness against model mismatch and track generalization.

This article proposes a trajectory-conditioned RL controller (TC-Driver) for resource-constrained hardware, inspired by the two-layer planner-controller separation that is often present in robotic systems [1], [12]. Within our framework, the planning layer is responsible for generating a safe and performant trajectory, while the control layer is dedicated to generating control inputs to make the system follow the given trajectory. According to our layout then, TC-Driver considers the planner to be given and uses the RL agent for trajectory tracking and velocity control, exploiting the learning capabilities of RL to heuristically handle model mismatch and track generalization, while leveraging the safety and reliability of traditional planning methods [31] in a hybrid fashion. The main contribution of this article is the design and evaluation of TC-Driver, a robust trajectory-conditioned RL approach for autonomous driving, specifically designed for resource-constrained hardware, such as an Intel Core i3-1115G4 central processing unit (CPU) or an NVIDIA Jetson NX. TC-Driver is able to effectively zero-shot transfer its driving behavior to an unseen track, as well as to robustly tackle varying tire conditions when compared with the SotA end-to-end RL architecture. Experimental results in simulation and on a physical F1TENTH race car [10] suggest that our hybrid architecture can zero-shot transfer on the physical system significantly better than the previous end-to-end architecture by demonstrating a tenfold lower crash ratio, which is computed as the proportion of laps that were not completed due to collision with boundaries to the total number of laps. Therefore, the TC-Driver architecture offers multiple advantages, as detailed.

A. ROBUSTNESS TO MODELING MISMATCH

Many previous works have highlighted the importance of model randomization when training RL agents in simulation for real-world application [28], [32]. However, previous SotA presents little [25], [28] to no [27] focus toward explicitly training autonomous racing algorithms in randomized settings and testing them in unseen circumstances. We specifically focus on this crucial theme for in-field driving, namely, by choosing the notoriously important tire parameters (e.g., tire friction and stiffness) [11], [17], [18], [19]. TC-Driver introduces the model randomization onto the tires’ friction to the RL agent during training, differently from previous works [25], [27], [28], as in Table 1, by injecting Gaussian noise varying at each episode throughout the training procedure. Experimental results specifically test the controller outside of the friction training domain, showing that the TC-Driver architecture brings a 29-fold crash ratio improvement when compared with an SotA end-to-end implementation and a 32-fold crash ratio improvement when compared with a nonlearning-based MPC, as in Table 2.

B. TRACK GENERALIZATION CAPABILITIES

Recent previous work on RL autonomous racing did not focus specifically on the agents’ ability to generalize to unforeseen tracks [25], [28], [33], rather they were interested in optimizing the control capabilities on the training track. On the contrary, this article shows that the proposed architecture can better generalize to unseen tracks, as the observation given to the RL model has no general reference to the track itself but only a partial trajectory. The TC-Driver yields superior generalization capabilities on unforeseen tracks when compared with the end-to-end setting based on previous Sota [25], [28], [33]. As shown in Table 3, the TC-Driver outperforms
end-to-end model by achieving an average crash ratio lower by a factor of ~2.5 in simulation and by a factor of 10 in reality. We can demonstrate similar track generalization characteristics as in [27], however, within a model-free setting.

C. COMPUTATIONAL BENEFIT

The best performing SotA controllers in autonomous racing are still MPC-based. However, they either require a powerful compute platform comparable to a desktop computer [1] or external compute [11] that is not always available under space and power consumption constraints, especially in racing competitions. To further motivate the adoption of RL agents for embedded autonomous driving, we evaluate the computational performance of our algorithm at runtime, showing that the RL inference has an average duration of 0.25 ms compared with the average MPC solving time of 11.5 ms, as in Table 4, allowing for deployment on either less performant platforms or at higher frequencies.

D. ZERO-SHOT Sim2Real CAPACITY

A recent survey [12] has highlighted that the majority of previous autonomous racing algorithms have not been proven to be working on real platforms, even less on a resource-constrained system. In fact, only 23 algorithms out of 49 are deployed on real platforms; only two of the 23 then are implemented on small form-factor, resource-constrained hardware. Focusing on RL algorithms then, multiple previous works only show their car working in simulation [25]. On the other hand, only the following works [27], [28] have demonstrated Sim2Real capabilities. This article presents and evaluates TC-Driver’s generalization performance on the physical F1TENTH system [10], showing that the proposed architecture possesses a great Sim2Real advantage compared with previous end-to-end RL architectures, by deploying RL models purely trained in simulation into the physical system on completely unforeseen tracks. TC-Driver outperforms the end-to-end setting tenfold in terms of crash ratio and is capable of demonstrating similar lap time consistency in reality as observed in simulation, as shown in Table 5. This is comparable to the Sim2Real capabilities demonstrated in [27], yet without the model being trained on the layout resembling the physical testing track (namely, [27] tested on the training track but in opposite direction), whereas this work emphasizes on a quantitative analysis of the lap-completion ratio on completely unforeseen tracks.

In summary, TC-Driver is a computationally efficient hybrid RL approach to autonomous racing that proves to be capable of robust control in terms of model parameter mismatch and track generalization, demonstrating lap completion under model mismatch settings where classical nonlearning-based MPC fails to do so and outperforming SotA end-to-end RL controllers ten fold in terms of crash ratio under zero-shot Sim2Real conditions. It, thus, demonstrates the viability and necessity of leveraging classical control strategies in the hybrid RL setting, as opposed to pure end-to-end architectures. A summary table of the features of TC-Driver compared with previous work is available in Table 1.

II. METHODOLOGY

The RL terminology follows the convention of [21]. The main goal of the proposed architecture is to train an agent operating a race car that is aware of a given trajectory in realistic conditions, especially under the influence of noise applied to the tire friction coefficients. As the environment will have different tire modeling parameters in every episode, the agent learns to handle the tire parameter modeling mismatch during training, ultimately allowing for robust tracking of a given trajectory. The method of presenting the RL agent with random environment dynamics, also called domain randomization, is often used in previous works [22], [28], [32] and often is deemed fundamental (e.g., [32]) to learn robust behavior in the face of model uncertainties that arise in real-world settings. The following sections present the proposed TC-Driver architecture starting from the simulation environment in Section II-A used to build and evaluate the proposed solution, as well as the SotA end-to-end architecture, in Section II-B. Finally, we describe the procedure of tire parameter randomization to ensure model mismatch robustness in Section II-C.

A. SIMULATION ENVIRONMENT

For a fast and accurate evaluation of the proposed architecture, we adopted the FITENTH simulation environment [9], which aims to offer an OpenAI-gym-compatible wrapper [34]. Within the environment, the vehicle’s dynamics are modeled with the Single Track model [35] to realistically simulate Ackermann-steered vehicles. The model can be seen in Fig. 1.

\[
\mu, \ C_{S_f}, \text{ and } C_{S_r} \text{ model the friction, the cornering stiffness on the front axle, and the cornering stiffness on the rear axle, respectively, as in [35] and [36]. The FITENTH environment has been modified to be able to inject noise into the simulation parameters, allowing the investigation of robustness in terms of tire modeling inaccuracies. The simulation environment offers the following dynamic state of the car: } \mathbf{q}_{\text{dyn}} = [x, y, \psi, v_x, v_y, \dot{\psi}] = [\text{global x position}, \text{global y position}, \text{yaw angle with respect to the positive x-axis, longitudinal velocity, lateral velocity, yaw rate}] \]
interaction, and hyperparameters. The used environment is the trajectory tracker, with their underlying architecture, environment interaction, and hyperparameters. The used environment is based on an adapted version of the FITENTH gym racing environment [9]. Both RL agents were implemented using the stable baselines 3 (SB3) soft actor critic (SAC) algorithm [37], which is an off-policy actor–critic deep RL algorithm that aims at maximizing the actor’s entropy together with the expected reward. SAC was initialized with $\gamma$ at 0.99, an episode length of 10,000, batch size of 64, train frequency of 1, and using the multilayer perceptron (MLP) policy.

1) END-TO-END RACER

To generate a baseline for comparisons, we utilized the frequently used model-free end-to-end architecture of [25], [28], and [33]. The chosen observation space recasts the original observation $\text{obs}_{\text{gym}}$ in a Frenet frame, which is a representation relative to a trajectory, as in [25], [28], and [33]: $\text{obs}_{\text{Frenet}} = [p, n, \dot{\psi}, v_x, v_y, \psi]$ = [progress along the path, perpendicular deviation from the path, relative heading, longitudinal velocity, lateral velocity, yaw rate]. An array of LiDAR distance measurements is also included; compared with the original scan, this is downsampled by taking only one every 108th scan, making the final scan$_{\text{filtered}}$ a ten-element array. The complete observation reads as follows: $\text{obs}_{\text{end2end}} = [\text{scan}_{\text{filtered}}, \text{obs}_{\text{Frenet}}]$. The final dimension of the observation space was 16, making the policy network a four-layer MLP with layer size (16, 256, 256, 2), respectively, and with a rectified linear unit (ReLU) activation function after the second and third layers. A schematic overview of the RL environment interaction with the end-to-end agent is visible in Fig. 3, as well as a diagram of the neural network (NN) in Fig. 4. The agent learns a control policy with online environment interaction based on the previously defined reward function in Section II-A. Since such advancement-based rewards were broadly tested [25], [27], [28], [33], we consider this agent a reasonable comparison model.

2) TRAJECTORY-CONDITIONED DRIVER

The proposed trajectory tracker TC- Driver tracks the spatial trajectory generated by a high-level planner. Within this work, a pregenerated model predictive contouring controller (MPCC) trajectory is used for training, which has been custom implemented for this task, following [11]. That is, the track has already been traversed by an MPC, and the logged trajectory can then be used by subsequent RL agents as the optimal tracking trajectory. It is worth mentioning that this trajectory could be chosen arbitrarily, such as, for example, by using the center-line trajectory instead of the time-optimal MPC trajectory. The observation space of the proposed trajectory tracker is slightly altered compared with the end-to-end setting. To enable trajectory following, we add a sample of the optimal trajectory relative to the current position of the car. This sample consists of 20 points taken at a 20-cm distance from each other, rotated, and translated to be in the car’s frame of reference. Therefore, the observation space in the spatial trajectory tracking setting is newly defined as $\text{obs}_{\text{traj}} = [\text{traj}, \text{obs}_{\text{end2end}}] = [\text{relative trajectory, scans, …}]

$$\dot{x} = v \cos(\psi + \beta)$$

$$\dot{y} = v \sin(\psi + \beta)$$

$$\delta = f_{\text{steer}}(x_3, u_1)$$

$$\psi = f_{\text{acc}}(x_4, u_2)$$

$$\psi = \frac{\mu}{l_s(l_r + l_f)} \left[ l_f C_s, f(g_{t_r} - u_2 h_{c0}) \delta 
+ (l_r C_s, r(g_{t_f} + u_2 h_{c0}) - l_f C_s, f(g_{t_r} + u_2 h_{c0})) \delta 
- (l_f^2 C_s, f(g_{t_r} - u_2 h_{c0}) + l_r^2 C_s, r(g_{t_f} + u_2 h_{c0})) \frac{\psi}{v} \right] \right)$$

$$\delta = \frac{\mu}{v(l_r + l_f)} \left[ C_s, f(g_{t_r} - u_2 h_{c0}) \delta 
- (C_s, r(g_{t_f} + u_2 h_{c0}) + C_s, f(g_{t_r} - u_2 h_{c0})) \delta 
+ (C_s, r(g_{t_f} + u_2 h_{c0}) l_r - C_s, f(g_{t_r} - u_2 h_{c0}) l_f) \frac{\psi}{v} \right] \left. \right] - \frac{\psi}{v}$$

**FIGURE 1.** Bicycle model dynamics from [35].

Furthermore, the simulation environment provides sensory input in the form of a LiDAR scan made of 1080 points over 270° coverage area around the car. In summary, the observation of the environment is $\text{obs}_{\text{gym}} = [\text{scan}, s_x, s_y, \psi, v_x, v_y, \dot{\psi}]$. The action space of the gym environment solely consists of continuous actions $a = [v, \delta]$, where $v$ is the desired longitudinal velocity and $\delta$ is the steering angle of the agent. The reward function is defined in the following equation inspired by [25], [28]:

$$r_t = \begin{cases} 
-c, & \text{if crashing} \\
\Delta \theta^t + p^\text{traj} \delta^t + p^\text{act} \delta^t, & \text{otherwise}.
\end{cases} \quad (1)$$

In the reward definition, $c = 1$, and $\Delta \theta^t$ is the track advancement at simulation time step $t$. $\delta^t$ is the deviation at time step $t$ from the previous action, measured as the two-norm of the difference between the two action vectors. $p^\text{act}$ is a tuning parameter that was also heuristically chosen to be 0.01. The reward is designed in a way to prime the parameters were chosen by validating a coarse choice of logarithmically spaced parameters and choosing the one that yielded the highest average advancement after a fixed training time. The training and test tracks can be seen in Fig. 2.

**B. RL ARCHITECTURES**

This section introduces both the frequently used end-to-end RL architecture [25], [26], [27], [28] and the RL trajectory tracker, with their underlying architecture, environment interaction, and hyperparameters. The used environment is
FIGURE 2. Training track $F$ depicted in blue. Testing tracks Autodrome, Catalunya, and Oschersleben, which are unseen during training. The tracks vary in length from 89 to 470 m. The centerline is depicted in gray.

FIGURE 3. RL environment structure. Both observation spaces are depicted in the picture with dashed borders. They are, however, mutually exclusive; only one at a time is used during training, and they define the two different agents, end-to-end and TC-Driver.

FIGURE 4. NN architectures for the two policy networks. End-to-end (top) and TC-Driver (bottom) architecture. The only point of difference between the architectures is the size of the input layer.

progress along the path, perpendicular deviation from the path, relative heading, longitudinal velocity, lateral velocity, yaw rate. The final dimension of the observation space was 56, making the policy network a four-layer MLP with layer size (56, 256, 256, 2), respectively, and with a ReLU activation function after the second and third layers. A diagram of the NN is also available in Fig. 4, and a schematic overview of the RL environment interaction with the TC-Driver agent is visible in Fig. 3. The reward function is as defined in Section II-A.

C. TIRE PARAMETER RANDOMIZATION
The FiTENTH simulation environment utilizes the single-track dynamic model of [35]. To apply randomness to the tire coefficients, Gaussian noise was applied at each reset of the gym environment during training. The noise was centered at the nominal friction value, used in the MPC to find the optimal trajectory. To determine the standard deviation, the limit of tire friction at which MPC would not be able to correctly complete a lap was analyzed. Then, the standard deviation of the noise was set to be half of that value for the noise to be mostly (but not entirely) inside the range of values that allow MPC to finish a lap. The numerical values are $\mu_{\text{noisy}} \sim N(1.0489, 0.0375)$.

III. EXPERIMENTAL RESULTS
This section evaluates the proposed trajectory tracking agent against the end-to-end agent with the tire parameter randomization during training. We, furthermore, compare the results of the ML-based agents with an MPC agent, which does not know the correct parameters, to simulate parameter mismatch. Evaluation metrics consist of lap time, the ability to handle different track conditions, and the ability to drive on unseen tracks. In simulation, the optimal trajectory of an MPC with exact model parameters (without tire noise, hence, with zero model mismatch) is used as the ground-truth reference. For all the results presented in this work, every agent was trained for $5 \cdot 10^5$ timesteps on the track named $F$, which can be seen in Fig. 2.

A. ROBUSTNESS TO TIRE MODELING MISMATCH
To test the capabilities of the algorithms to generalize to different tire frictions, 200 randomly extracted values were utilized during test laps. To better test the generalization capabilities, these friction parameters were extracted in an interval that was predominantly outside the training range. Namely, the normal distribution had a mean 0.2 lower than the nominal one, with the same standard deviation as in the training phase, i.e., 0.0375, thus making the track considerably more slippery. The MPC was run with the nominal system model, i.e., the tires’ friction was not changed, to simulate model mismatch. The three different models were run on the track, starting from the same position, for one lap. In Fig. 5, one can see a trajectory extract, with the 200 laps superimposed one on the other.
FIGURE 5. Agents that were trained under tire friction randomization within the MPC tolerance are tested in an environment outside of the trained tire friction domain. End-to-end agent (left), proposed TC-Driver (middle), and MPC (right). Crosses are used to indicate crashes into the track walls; as it can be seen, only the TC-Driver manages to drive in the shown chicane. Models were tested for 200 runs on the training track $F$.

Due to parameter mismatch, MPC suffers an 80.50% crash ratio; the domain-randomization-aware end-to-end agent instead yields a 73.50% crash ratio. TC-Driver heavily outperforms both methods, with a crash ratio of only 2.50%, improving by a factor of $\sim 32$ on the result of MPC and by a factor of $\sim 29$ on the result of the end-to-end agent. This increase in robustness comes with only a marginally lower lap time when compared with MPC: TC-Driver is only $\sim 7\%$ slower, with a lap time of 10.798 s opposed to 10.094 s of the MPC. When TC-Driver is compared with the end-to-end agent instead, it turns out to be faster, as the end-to-end agent has an average lap time of 11.148 s. The lower lap time of MPC should not, however, be considered as higher performance, as the excessive amount of crashing makes it a nonsuitable controller in this setting. Especially, if the average lap completion across the experiments is taken into account, it is clear that TC-Driver is the only controller robust enough in this situation: it completes on average 99.37% of the lap, while the end-to-end method only completes 52.52% on average, and the MPC only 32.67%, showing that they have consistent problems with this amount of model mismatch.

Regarding the MPC, it has to be said that such a high crash ratio is expected, as the tire mismatch is purposely chosen to make it fail. A solution for such a situation would be the integration of learnable parameters within the MPC model, as in [19]. Hence, this result does not exhibit superiority to the general class of MPC but rather demonstrates a case in which RL can be utilized in the mitigation of model mismatch.

**B. TRACK GENERALIZATION CAP ABILITIES**

To test the trajectory-conditioned driver’s ability to generalize to race tracks beyond the training track of $F$, it was evaluated on three additional unforeseen tracks, namely, Autodrome, Catalunya, and Oschersleben, without tire parameter randomization, as visible in Fig. 2. The tracks were obtained from the open-source repository accompanying [38]. The agents were started at 200 different positions along these tracks and drove a single lap each. To further emphasize the generalization capabilities to arbitrary trajectories, the trajectories used for conditioning the TC-Driver were obtained from the minimum curvature optimizer shown in [38], instead of the MPCC traversed trajectory as utilized during training.

Table 3 depicts the described runs on the unseen tracks visible in Fig. 6. The MPC clearly outperforms both RL methods, as expected in a zero model-mismatch setting, where the optimality within the receding horizon holds. It shows the fastest lap times on all test tracks with the lowest standard deviation, while never crashing, as it has perfect model knowledge. Regarding the RL agents, we notice that the end-to-end agent is not able to generalize to the different new tracks effectively, never managing to complete more than 5% of the laps. Specifically, on the Oschersleben track, it never manages to complete a lap. On the other hand, the TC-Driver manages to successfully generalize to Autodrome and to complete more than 40% of the laps on Catalunya. It only struggles to complete Oschersleben, achieving lap completion only 6% of the time. Looking at the average advancement comparison,

|                 | $t_\mu$ [s] | $t_\sigma$ [s] | Crashes [%] | adv$_\mu$ [%] | adv$_\sigma$ [%] |
|----------------|-------------|---------------|-------------|---------------|-----------------|
| MPC            | 10.094      | 0.501         | 80.50%      | 32.67%        | 28.26%          |
| end-to-end     | 11.148      | 0.302         | 73.50%      | 52.51%        | 28.69%          |
| TC-Driver      | 10.798      | 0.143         | 2.50%       | 99.37%        | 4.90%           |
we notice that the TC-Driver outperforms the other RL agent in all cases. On the worst performing track Oschersleben, TC-Driver still completes more than twice the distance of the end-to-end driver, and in the best case, our proposed agent completes on average 2.7 more lap length, in the circuit called Autodrome.

Looking at the lap times, we see that the end-to-end agent achieves significantly lower lap times, displaying aggressive behavior. This characteristic of the controller causes the end-to-end agent to crash frequently, not allowing it to complete a single lap and, therefore, does not demonstrate robustness to track generalization. We argue that the main reason for crashing could be the particularly different features present in some of the testing tracks. We used three real-world downscaled tracks, that all present some features, which are not specifically present in the F training track. A specific feature that is not present in the training track F is that of a high-speed chicane, which consists of a fast left and subsequent right turn (or vice versa), and it can be seen in the last rows of Fig. 6, as a part of the track Oschersleben. Here, we can see two of the high-speed chicanes, and we can see that the TC-Driver also occasionally fails at driving in this situation.

C. COMPUTATION TIME

We focus on the computational time of the utilized control methods. Table 4 depicts the average computation time of each method and their respective standard deviation. The following computations were performed on an Intel i7-10700K CPU. The MPC’s average computation duration is approximately 11 ms with a rather high standard deviation of 0.9 ms. The reason for the higher deviation arises from the nature of quadratic programming, which is subject to constantly varying solving conditions. On the other hand, both RL algorithms show a significantly lower and more constant inference time of approximately 0.26 ms. Thus, the RL computation time is faster by a factor of roughly 40, showing the potential of ML to bring high-performance and robust control methods to resource-constrained embedded hardware.

D. Sim2Real CAPACITIES

To investigate and validate the simulation results of the proposed architecture, the Sim2Real capability of the TC-Driver is demonstrated on a physical racecar in the 1:10 form factor, namely, the F1TENTH platform [10]. The robot is built upon the off-the-shelf Traxxas 4 × 4 Slash race chassis.
power train, driven by a VESC 6 MkIV electronic speed controller (ESC). Furthermore, the robot sensors consist of the integrated inertial measurement unit (IMU) of the ESC as well as a Hokuyo UST-10LX laser range measurement sensor. The onboard computer is an NUC10i3FNKNI3-10110U running standard Ubuntu 20.04 and robot operating system (ROS) Noetic. A state estimator and a trajectory planner have been implemented, capable of emulating both previously introduced observation spaces \(\text{obs}_{\text{traj}}\) and \(\text{obs}_{\text{end2end}}\). The state estimator consists of a simultaneous localization and mapping (SLAM) algorithm based on [39] for positional estimates and an extended Kalman filter (EKF) based on [40] for velocity estimates, ultimately allowing for the estimation of the complete dynamic state as in [35] and [36]. The trajectory planner computes a globally optimal trajectory of a given track, with respect to minimum curvature, based on [38], which is the same planner as used previously in Section III-B. Finally, an ROS wrapper allows feeding the proper observation to the RL model, which, in turn, infers the actions for the actuators of the robot. Our hardware platform and the race-track setup are shown in Fig. 7.

1) ZERO-SHOT Sim2Real TRACK GENERALIZATION
Both the end-to-end and the TC-Driver RL agents are deployed on a physical racetrack outside of their training distribution without any model refinement. Thus, the agents have to demonstrate zero-shot Sim2Real capabilities directly out of the simulator environment to the physical system, as we quantify their respective performance in terms of lap time and crash ratio, by repeating the runs ten times each in both clockwise and counterclockwise directions. Both the deployed agents have been trained purely in simulation and for the same duration as in Section III. They also have the same action space and run on the same physical platform on the same track, thus serving fair comparison conditions.

![FIGURE 8. Physical zero-shot generalization run of the end-to-end and TC-driver algorithms (from left to right), on an unseen track. Clockwise runs were repeated ten times each.](image)

As can be seen in Fig. 8, the proposed TC-Driver yields superior zero-shot generalization capabilities when compared with the end-to-end setting. This coincides with the results of the simulation environment in Table 3. TC-Driver tracks the optimal race line significantly closer, as well as with lower variance than the end-to-end agent. As is visible from Table 5, the TC-Driver outperforms the end-to-end architecture in terms of crash ratio, by only crashing twice, thus resulting in a 10% crash ratio and a mean lap time \(t_\mu\) of 20.281 s with a very constant lap time standard deviation \(t_\sigma\) of 0.371 s, indicating a deterministic behavior. Interestingly to mention is that the TC-Driver manages to retain similar metrics in terms of lap time standard deviation \(t_\sigma\) and crash ratio, as achieved in Table 3, on a completely different track. The end-to-end agent, on the other hand, is not able to perform a single lap without crashing; thus, both lap time \(t_\mu\) and the respective standard deviation \(t_\sigma\) do not yield a measurable value.

| TABLE 5. Sim2Real lap time results of ten runs in a clockwise direction and ten runs in a counterclockwise direction, of end-to-end and TC-Driver RL architectures on the physical track. Average lap time \(t_\mu\) in seconds (lower is better). Standard deviation of the lap times \(t_\sigma\) (lower is better). Percentage of crashes during the runs (lower is better). |
| --- | --- | --- |
| Lap time \(t_\mu\) [s] | Lap time \(t_\sigma\) [s] | Crashes |
| end-to-end | n.a. | n.a. | 100.0% |
| TC-Driver | 20.281 | 0.373 | 10.0 % |

IV. CONCLUSION
This article presented TC-Driver, a hybrid RL approach to autonomous racing, that leverages the heuristic nature of RL and the reliability of traditional planning techniques. Given the imperfect modeling of parameters, MPC’s optimality
does not hold, leading to slower lap times and potentially even crashes. RL offers a viable approach to this solution by generalizing to different driving conditions. Yet, end-to-end RL methods rely on states that are not fit for efficient generalization to different tracks nor to model mismatch. Combining a traditionally generated trajectory in an observation for an RL agent tracking the trajectory under changing driving conditions alleviates these shortcomings. We evaluated and compared these approaches both in the simulated F1TENTH autonomous racing environment [9] as well as on the physical F1TENTH platform [10]. The proposed TC-Driver architecture shows that it can adapt to model mismatch scenarios that a non-learning-based MPC fails to handle [17]. It achieves lower and more consistent lap times, compared with the end-to-end agent based on [25], [28], and [33], and has by far the lowest overall crash ratio in the model mismatch setting (MPC: 80.50%, end-to-end: 73.50%, and TC-Driver: 2.50%). Furthermore, when deployed on test tracks that have significantly different features than the training track, our agent is capable of completing laps, demonstrating zero-shot track generalization capabilities, unlike previous end-to-end architectures (crash ratio in Autodrome track; end-to-end: 96% and TC-Driver: 8%). Finally, experimental results demonstrate zero-shot Sim2Real generalization capabilities on a custom-built racing platform and track. The physical test also yields similar consistency metrics as in simulation in terms of lap time deviation with $t_{avg} \sim 0.373$ s and displays a tenfold lower crash ratio than the end-to-end agent in a zero-shot Sim2Real setting.

Future work on this topic regards the alleviation of the bang–bang control characteristics that are in the nature of the RL architecture. This could potentially be mitigated by introducing output regularization such as in [28] or the introduction of Bernoulli policies [41]. Finally, a highly interesting RL approach would be the utilization of a model-based RL architecture as well as the integration of a recurrent NN (RNN) architecture, as inspired by [27]. As the computational effort required by ML techniques is greatly inferior to the effort required for optimal control techniques ($\sim 40 \times$ in our case), we deem this a promising line of work for bringing high-performance racing algorithms on real hardware-constrained platforms. The code for reproducing all mentioned RL and MPC F1TENTH implementations, as well as further result material, is available at https://github.com/ETH-PBL/TC-Driver.

ACKNOWLEDGMENT
The authors would like to thank Dr. Christian Vogt, Dr. Andrea Carron, and Dr. Alexander Liniger of ETH Zürich, for their constructive and fruitful algorithmic discussions, and Dr. Niao He, whose RL lecture project at ETH enabled the initial stepping stone to this work.

REFERENCES
[1] J. Kabzan et al., “AMZ driverless: The full autonomous racing system,” J. Field Robot., vol. 37, no. 7, pp. 1267–1294, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21977
[2] C. Lyu et al., “Toward a gliding hybrid aerial underwater vehicle: Design, fabrication, and experiments,” J. Field Robot., vol. 39, no. 5, pp. 543–556, Aug. 2022.
[3] J. Tuning, S. Cho, D. Lee, H. Lee, and D. H. Shim, “A direct visual servoing-based framework for the 2016 IROS autonomous drone racing challenge,” J. Field Robot., vol. 35, no. 1, pp. 146–166, Jan. 2018.
[4] C. K. Law, D. Dalal, and S. Shearow, “Robust model predictive control for autonomous vehicles/self-driving cars,” 2018, arXiv:1805.08551.
[5] U. Rosolia and F. Borrelli, “Learning how to autonomously race a car: A predictive control approach,” 2019, arXiv:1901.08184.
[6] T. J. Crayton and B. M. Meier, “Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy,” J. Transp. Health, vol. 6, pp. 245–252, Sep. 2017.
[7] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” IEEE Access, vol. 8, pp. 58443–58469, 2020.
[8] B. Balaji et al., “DeepRacer: Educational autonomous racing platform for experimentation with Sim2Real reinforcement learning,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), Paris, France, 2020, pp. 2746–2754, doi: 10.1109/ICRA40945.2020.9197465.
[9] M. O’Kelly, H. Zheng, D. Karthik, and R. Mangharam, “F1TENTH: An open-source evaluation environment for continuous control and reinforcement learning,” in Proc. NeurIPS Competition Demonstration Track, 2020, pp. 77–89.
[10] M. O’Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, “TUNERCAR: A superoptimization toolchain for autonomous racing,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2020, pp. 5356–5362.
[11] A. Liniger, A. Domahidi, and M. Morari, “Optimization-based autonomous racing of 1:43 scale RC cars,” Optim. Control Appl. Methods, vol. 36, no. 5, pp. 628–647, Jul. 2014, doi: 10.1002/oca.2123.
[12] J. Betz et al., “Autonomous vehicles on the edge: A survey on autonomous vehicle racing,” 2022, arXiv:2202.07008.
[13] Y. Wang, S. G. Advani, and A. K. Prasad, “A comparison of rule-based and model predictive controller-based power management strategies for fuel cell/battery hybrid vehicles considering degradation,” Int. J. Hydrogen Energy, vol. 45, no. 58, pp. 33948–33956, Nov. 2020, doi: 10.1016/j.ijhydene.2020.09.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360319920334194
[14] H. B. Pacejka, “Tire characteristics and vehicle handling and stability,” in Tire and Vehicle Dynamics, 3rd ed., Oxford, U.K.: Butterworth-Heinemann, 2012, ch. 1, pp. 1–58. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080970165000012
[15] A. Raji et al., “Motion planning and control for multi vehicle autonomous racing at high speeds,” in Proc. 25th Int. Conf. Intell. Transp. Syst., Oct. 2022, pp. 2775–2782.
[16] M. Brown and J. C. Gerdes, “Coordinating tire forces to avoid obstacles using nonlinear model predictive control,” IEEE Trans. Intell. Vehicles, vol. 5, no. 1, pp. 21–31, Mar. 2020.
[17] A. Liniger, “Pushing the limits of friction: A story of model mismatch,” in Proc. ICRA Autonomous Racing, 2021, pp. 1–30. [Online]. Available: https://www.youtube.com/watch?v=rTawyZghEg&t=136s
[18] L. P. Fröhlich, C. Kuttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron, “Model learning and contextual controller tuning for autonomous racing,” 2021, arXiv:2110.02710.
[19] A. Jain, M. O’Kelly, P. Chaudhari, and M. Morari, “Bayesrace: Learning to race autonomously using prior experience,” in Proc. Conf. Robot Learrn., vol. 155, J. Kober, F. Ramos, and C. Tomlin, Eds., 2021, pp. 1918–1929. [Online]. Available: https://proceedings.mlr.press/v155/jain21h.html
[20] J. V. Carrau, A. Liniger, X. Zhang, and J. Lygeros, “Efficient implementation of randomized mpc for miniature race cars,” in Proc. 15th Eur. Control Conf. (ECC), Aalborg, Denmark. Piscataway, NJ, USA: IEEE, Jun. 2016, pp. 957–962.
[21] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html
[22] J. Degrave et al., “Magnetic control of tokamak plasmas through deep reinforcement learning,” Nature, vol. 602, pp. 414–419, Feb. 2022.
[23] O. M. Andrzejowicz et al., “Learning dexterous in-hand manipulation,” Int. J. Robot. Res., vol. 39, no. 1, pp. 3–20, 2020.

VOLUME 1, 2024

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, P. Polack, F. Altché, B. d’Andréa-Novel, and A. de La Fortelle, “The M. Althoff, M. Koschi, and S. Manzinger, “CommonRoad: Composable G. Brockman et al., “OpenAI gym,” 2016, arXiv:1606.01540 Y. Song, H. Lin, E. Kaufmann, P. Dürr, and D. Scaramuzza, “Autonomous overtaking in Gran Turismo sport using curriculum reinforcement learning,” in Proc. IEEE Int. Conf. Robot. Autom. Lett., vol. 3, no. 3, pp. 1394–1401, Jul. 2018. D. Li, D. Zhao, Q. Zhang, and Y. Chen, “Reinforcement learning and deep learning based lateral control for autonomous driving,” 2018, arXiv:1810.12778. M. Werling, J. Ziegler, S. Kammel, and S. Thrun, “Optimal trajectory generation for dynamic street scenarios in a French frame,” in Proc. IEEE Int. Conf. Robot. Autom., May 2010, pp. 987–993. A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, “Deep drone racing: From simulation to reality with domain randomization,” IEEE Trans. Robot., vol. 36, no. 1, pp. 1–14, Feb. 2020. Y. Song, H. Lin, E. Kaufmann, P. Dürr, and D. Scaramuzza, “Autonomous overtaking in Gran Turismo sport using curriculum reinforcement learning,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2021, pp. 9403–9409. G. Brockman et al., “OpenAI gym,” 2016, arXiv:1606.01540. M. Althoff, M. Koschi, and S. Manzinger, “CommonRoad: Composable benchmarks for motion planning on roads,” in Proc. IEEE Intell. Vehicles Symp. (IV), Los Angeles, CA, USA, Jun. 2017, pp. 719–726. [Online]. Available: http://ieeexplore.ieee.org/document/7993802. P. Polack, F. Altché, B. d’Andréa-Novell, and A. de La Fortelle, “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” in Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2017, pp. 812–818. T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft-actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” 2018, arXiv:1801.01290. A. Heilmeier, A. Wochnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, “Minimum curvature trajectory planning and control for an autonomous race car,” Vehicle Syst. Dyn., vol. 58, no. 10, pp. 1497–1527, Oct. 2020. W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D LiDAR SLAM,” in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), May 2016, pp. 1271–1278. T. Moore and D. Stouch, “A generalized extended Kalman filter implementation for the robot operating system,” in Proc. 13th Int. Conf. Intell. Auton. Syst. (IAS-13). Cham, Switzerland: Springer, Jul. 2014, pp. 335–348. T. Seyde et al., “Is bang-bang control all you need? Solving continuous control with Bernoulli policies,” 2021, arXiv:2111.02552.