TC-Driver: A Trajectory Conditioned Reinforcement Learning Approach to Zero-Shot Autonomous Racing

Edoardo Ghignone*, Nicolas Baumann* and Michele Magno*
ETH Zurich, Switzerland

Abstract: Autonomous racing is becoming popular for academic and industry researchers as a test for general autonomous driving by pushing perception, planning, and control algorithms to their limits. While traditional control methods such as model predictive control are capable of generating an optimal control sequence at the edge of the vehicles’ physical controllability, these methods are sensitive to the accuracy of the modeling parameters, such as tire modeling coefficients. As model mismatch is inevitable in reality, the heuristic nature of Reinforcement Learning (RL) offers a viable approach to modeling robustness. This paper presents TC-Driver, an RL approach for robust control in autonomous racing. In particular, the TC-Driver agent is conditioned by a trajectory generated by any arbitrary traditional high-level trajectory planner. The proposed TC-Driver architecture addresses the tire parameter modeling inaccuracies by exploiting the learning capabilities of RL while utilizing the reliability of traditional planning methods in a hybrid fashion. We train the agent under varying tire conditions, allowing it to generalize to different model parameters, aiming to increase the racing capabilities of the system in practice. Experimental results demonstrate that the proposed hybrid RL architecture of the TC-Driver improves the generalization robustness of autonomous racing agents when compared to a previous state-of-the-art end-to-end-based architecture. Namely, the proposed controller yields a 29-fold improvement in crash ratio when facing model mismatch and can zero-shot transfer its behavior on unseen tracks which present completely new features, while the end-to-end baseline fails. When deployed on a physical system, the proposed architecture demonstrates zero-shot Sim2Real capabilities that outperform end-to-end agents 10-fold in terms of crash ratio while exhibiting similar driving characteristics in reality as in simulation.

Keywords: autonomous racing, reinforcement learning, control, wheeled robots, embedded control

1. Introduction
Autonomous racing on resource-constrained hardware pushes the boundaries of algorithmic design and implementation in perception, planning, and control (Jung et al., 2018; Kabzan et al., 2020;
Thus, it is a valuable asset for researchers to push the limits of autonomous driving (Law et al., 2018; Rosolia and Borrelli, 2020), as it can lead to many benefits, such as enhancing road safety, reducing carbon emissions, transporting the mobility impaired, and reducing driving-related stress (Crayton and Meier, 2017; Yurtsever et al., 2020). 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 (Kabzan et al., 2020) and the AWS DeepRacer League (Balaji et al., 2020) are popular and followed by tens of teams. Among other competitions, the FITENTH racing platform (O’Kelly et al., 2020b) is gaining popularity by organizing semi-regular 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 (O’Kelly et al., 2020a), 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 (Betz et al., 2022; Liniger et al., 2014). Current State-of-the-Art (SotA) racing controllers utilize optimal control methods such as Model Predictive Control (MPC) (Kabzan et al., 2020; Law et al., 2018; Liniger et al., 2014; Rosolia and Borrelli, 2020). 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 Wang et al. (2020), heuristic strategies can outperform MPC even if the latter contains more information about the controlled system. Particularly in the context of autonomous car racing, the model inaccuracies of the lateral tire forces are highly critical (Facejka, 2012; Raji et al., 2022). These forces are notoriously difficult to model and the tires’ behavior is highly nonlinear (Brown and Gerdes, 2020; Liniger, 2021). In real racing scenarios, a tire modeling mismatch is very likely to occur, as high wear, tear, and weight changes modify the initial parameters (Liniger, 2021). While there exist several previous works that have attempted to address this issue using learning-based methods (Carrau et al., 2016; Fröhlich et al., 2021; Jain et al., 2021) for MPC, we assess and highlight the feasibility and performance of a reinforcement learning (RL) approach that allows robust control behavior without the need for complex model-contextual optimization in MPC.

RL (Sutton and Barto, 2018) methods offer a Machine Learning (ML)-based solution that was shown to be able to handle complex robotic and control tasks, such as plasma control (Degrave et al., 2022), hand manipulation (Andrychowicz et al., 2020), quadrupedal locomotion (Miki et al., 2022), and autonomous racing as in Fuchs et al. (2021) and Wurman et al. (2022), where the authors apply RL to outperform professional human drivers in the setting of a highly realistic videogame. In Brunbauer et al. (2022), instead the authors 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. In Chisari et al. (2021) then, the authors 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 (de Bruin et al., 2018; Li et al., 2018) 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 in de Bruin et al. (2018) where the authors show that using an auto-encoder 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 paper 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 (Betz et al., 2022; Kabzan et al., 2020). 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 utilizing the safety and reliability of traditional planning methods (Werling et al., 2010) in a hybrid fashion. The main contribution of this paper 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 to the SotA end-to-end RL architecture. Experimental results in simulation and on a physical F1TENTH race car (O’Kelly et al., 2020a) 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 10-fold 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 the following multiple advantages.

**Robustness to Modeling Mismatch:** Many previous works have highlighted the importance of model randomization when training RL agents in simulation for real-world application (Chisari et al., 2021; Loquercio et al., 2020). However, previous SotA presents little (Chisari et al., 2021; Fuchs et al., 2021) to no (Brunnbauer et al., 2022) focus towards 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) (Fröhlich et al., 2021; Jain et al., 2021; Liniger, 2021; Liniger et al., 2014). TC-Driver introduces the model randomization onto the tires’ friction to the RL agent during training, differently from previous works (Brunnbauer et al., 2022; Chisari et al., 2021; Fuchs et al., 2021), 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 to a SotA end-to-end implementation, and a 32-fold crash ratio improvement when compared to a non-learning-based MPC, as in Table 2.

**Track Generalization Capabilities:** Recent previous work on RL autonomous racing did not focus specifically on the agents’ ability to generalize to unforeseen tracks (Chisari et al., 2021; Fuchs et al., 2021; Song et al., 2021); rather they were interested in optimizing the control capabilities on the training track. On the contrary, this paper shows that the proposed architecture

---

**Table 1.** Comparison of related works in the field of RL autonomous racing. The • denotes partial investigation. This work studies the effect of Sim2Real, track generalization, and model generalization, while not relying on a fully black-box end-to-end method.

<table>
<thead>
<tr>
<th></th>
<th>Sim2Real</th>
<th>End-to-End</th>
<th>Track Generalization</th>
<th>Model Generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chisari et al. (2021)</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Brunnbauer et al. (2022)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fuchs et al. (2021)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ours</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table 2.** Lap time results of 200 runs; comparison with imperfect knowledge of dynamics on the F training track. Average lap time $t_{l}$ in seconds (lower is better, only completed laps are counted); standard deviation of the lap times $t_{l}$ (lower is better); percentage of crashes during the runs (lower is better). Average advancement $adv_{v}$ on the track per run, as a percentage of the complete laps (higher is better, all laps are counted); standard deviation of the advancement $adv_{v}$ on the track per run.

<table>
<thead>
<tr>
<th></th>
<th>Lap time, $t_{l}$ [s]</th>
<th>Lap time, $t_{l}$ [s]</th>
<th>Crashes</th>
<th>Advancement, $adv_{v}$ [%]</th>
<th>Advancement, $adv_{v}$ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC</td>
<td>10.094</td>
<td>0.051</td>
<td>80.50%</td>
<td>32.67%</td>
<td>28.26%</td>
</tr>
<tr>
<td>End-to-end</td>
<td>11.148</td>
<td>0.030</td>
<td>73.50%</td>
<td>52.51%</td>
<td>28.69%</td>
</tr>
<tr>
<td>TC-Driver</td>
<td>10.798</td>
<td>0.143</td>
<td>2.50%</td>
<td>99.37%</td>
<td>4.90%</td>
</tr>
</tbody>
</table>
Table 3. Averaged lap time results of 200 runs on the unseen tracks Autodrome, Catalunya, and Oschersleben with zero model mismatch. Average lap time in seconds (lower is better); standard deviation of the lap times (lower is better); percentage of crashes during the runs (lower is better); average advancement $adv_v$ on the track per run, as a percentage of the complete lap (higher is better, all laps are counted); standard deviation of the advancement $adv_v$ on the track per run (lower is better). The main comparison concerns the RL agents, as the MPC is in a zero-model-mismatch setting where its optimality holds, yet its performance is shown in gray for reference.

<table>
<thead>
<tr>
<th>Track</th>
<th>Driver</th>
<th>Lap time, $t_\mu$ [s]</th>
<th>Lap time, $t_\sigma$ [s]</th>
<th>Crashes</th>
<th>Advancement, $adv_v$ $\mu$</th>
<th>Advancement, $adv_v$ $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autodrome</td>
<td>MPC</td>
<td>46.461</td>
<td>0.029</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>End-to-end</td>
<td>52.557</td>
<td>0.234</td>
<td>96.00%</td>
<td>35.09%</td>
<td>27.06%</td>
</tr>
<tr>
<td></td>
<td>TC-Driver</td>
<td>59.020</td>
<td>0.307</td>
<td>8.00%</td>
<td>95.32%</td>
<td>17.88%</td>
</tr>
<tr>
<td>Catalunya</td>
<td>MPC</td>
<td>41.475</td>
<td>0.036</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>End-to-end</td>
<td>46.878</td>
<td>0.207</td>
<td>95.50%</td>
<td>44.16%</td>
<td>30.33%</td>
</tr>
<tr>
<td></td>
<td>TC-Driver</td>
<td>52.978</td>
<td>0.321</td>
<td>59.50%</td>
<td>65.27%</td>
<td>37.03%</td>
</tr>
<tr>
<td>Oschersleben</td>
<td>MPC</td>
<td>25.915</td>
<td>0.022</td>
<td>0.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>End-to-end</td>
<td>n.a.</td>
<td>n.a.</td>
<td>100.00%</td>
<td>19.27%</td>
<td>19.93%</td>
</tr>
<tr>
<td></td>
<td>TC-Driver</td>
<td>34.603</td>
<td>0.415</td>
<td>94.00%</td>
<td>46.95%</td>
<td>31.23%</td>
</tr>
</tbody>
</table>

Table 4. Average computation time of the utilized control methods and their respective standard deviation.

<table>
<thead>
<tr>
<th>Computation Time, $t_\mu$ [ms]</th>
<th>Computation Time, $t_\sigma$ [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC 11.2</td>
<td>0.9</td>
</tr>
<tr>
<td>End-to-end 0.26</td>
<td>0.05</td>
</tr>
<tr>
<td>TC-Driver 0.27</td>
<td>0.04</td>
</tr>
</tbody>
</table>

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. TC-Driver yields superior generalization capabilities on unforeseen tracks when compared to the end-to-end setting based on previous SotA implementations (Chisari et al., 2021; Fuchs et al., 2021; Song et al., 2021). As shown in Table 3, TC-Driver outperforms the end-to-end model by achieving an average crash ratio lower by a factor of $\sim 2.5$ in simulation and by a factor of 10 in reality. We can demonstrate similar track generalization characteristics as in Brunnbauer et al. (2022), however, within a model-free setting.

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 (Kabzan et al., 2020) or external compute (Liniger et al., 2014) 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 to 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.

Zero-Shot Sim2Real Capacity: A recent survey (Betz et al., 2022) has highlighted that the majority of previous autonomous racing algorithms have not been proven to be working on real platforms, and even less on a resource-constrained system. In fact, only 23 algorithms out of 49 are deployed on real platforms; only 2 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 (Fuchs et al., 2021). On the other hand, only Chisari et al. (2021) and Brunnbauer et al. (2022) have demonstrated Sim2Real capabilities. This paper presents and evaluates TC-Driver’s generalization performance on the physical F1TENTH system (O’Kelly et al., 2020a), showing that the proposed architecture possesses a great Sim2Real advantage compared to 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 10-fold 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
Table 5. Sim2Real lap time results of 10 runs in a clockwise direction and 10 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).

<table>
<thead>
<tr>
<th></th>
<th>Lap time, $t_\mu$ [s]</th>
<th>Lap time, $t_\sigma$ [s]</th>
<th>Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-to-end</td>
<td>n.a.</td>
<td>n.a.</td>
<td>100.0%</td>
</tr>
<tr>
<td>TC-Driver</td>
<td>20.281</td>
<td>0.373</td>
<td>10.0%</td>
</tr>
</tbody>
</table>

to the Sim2Real capabilities demonstrated in Brunnbauer et al. (2022), yet without the model being trained on the layout resembling the physical testing track (namely, Brunnbauer et al. (2022) tested on the training track but in the opposite direction), whereas this work emphasizes on a quantitative analysis of the lap-completion ratio on completely unforeseen tracks.

To summarize, 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 non-learning-based MPCs fail to do so and outperforming SotA end-to-end RL controllers 10-fold in terms of crash ratio under zero-shot Sim2Real conditions. It thus demonstrates the viability and necessity of utilizing classical control strategies in the hybrid RL setting, as opposed to pure end-to-end architectures. A summary of the features of TC-Driver compared to previous work is available in Table 1.

2. Methodology

The RL terminology follows the convention of Sutton and Barto (2018). 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 (Degrave et al., 2022; Loquercio et al., 2020; Chisari et al., 2021) and often is deemed fundamental (e.g., Loquercio et al., 2020) to learn robust behavior in the face of model uncertainties that arise in real-world settings. The following subsection presents the proposed TC-Driver architecture starting from the simulation environment in Section 2.1 used to build and evaluate the proposed solution, as well as the SotA end-to-end architecture in Section 2.2. Lastly, we describe the procedure of tire parameter randomization to ensure model mismatch robustness in Section 2.3.

2.1. Simulation Environment

For a fast and accurate evaluation of the proposed architecture, we adopted the F1TENTH simulation environment (O’Kelly et al., 2020b), which aims to offer an OpenAI-Gym-compatible wrapper (Brockman et al., 2016). Within the environment, the vehicle’s dynamics are modeled with the Single Track model (Althoff et al., 2017) to realistically simulate Ackermann-steered vehicles. The model can be seen in Figure 1.

$\mu$, $C_{S,f}$, and $C_{S,r}$ model the friction, the cornering stiffness on the front axle, and the cornering stiffness on the rear axle, respectively, as in Polack et al. (2017) and Althoff et al. (2017). The F1TENTH 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: $s_{dyn} = [s_x, s_y, \psi, v_x, v_y, \dot{\psi}] = [\text{global } x \text{ position, global } y \text{ position, yaw angle with respect to the positive } x \text{ axis, longitudinal velocity,}]$.
\[
\begin{align*}
\dot{s}_x &= v \cos(\psi + \beta) \\
\dot{s}_y &= v \sin(\psi + \beta) \\
\dot{\delta} &= f_{\text{steer}}(x_3, u_1) \\
\dot{v} &= f_{\text{acc}}(x_4, u_2) \\
\dot{\psi} &= \mu_m I_z (l_r + l_f) (C_{S,r}(gl_f - u_2 h_{cg}) - l_f C_{S,f}(gl_r + u_2 h_{cg})) \beta \\
\dot{\beta} &= \mu v (l_r + l_f) (C_{S,f}(gl_r - u_2 h_{cg}) - (C_{S,r}(gl_f + u_2 h_{cg}) + C_{S,f}(gl_r - u_2 h_{cg})) \beta \\
&\quad + (C_{S,r}(gl_f + u_2 h_{cg}) l_r - C_{S,f}(gl_r - u_2 h_{cg}) l_f) \frac{\dot{\psi}}{v} - \dot{\psi}
\end{align*}
\]

Figure 1. Bicycle model dynamics from Althoff et al. (2017).

lateral velocity, yaw rate]. 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. To summarize, 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 defined in Equation 1 is inspired by Chisari et al. (2021) and Fuchs et al. (2021):

\[
r_t = \begin{cases} 
-c \\
\Delta \theta_t + p^{\text{traj}} \delta_{t}^{\text{traj}} + p^{\text{act}} \delta_{t}^{\text{act}} \\
\text{if crashing} \\
\text{otherwise.}
\end{cases}
\]

In the reward definition \(c = 1\) and \(\Delta \theta_t\) is the track advancement at simulation time step \(t\). \(\delta_{t}^{\text{traj}}\) indicates the distance to the optimal trajectory at time step \(t\), and \(p^{\text{traj}}\) is a scaling parameter, which was heuristically set to be 0.05. \(\delta_{t}^{\text{act}}\) is the deviation at time step \(t\) from the previous action, measured as the 2-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 agent towards the optimal trajectory. The specific values of 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 Figure 2.

2.2. Reinforcement Learning Architectures

This section introduces both the frequently used end-to-end RL architecture (Brunnbauer et al., 2022; Chisari et al., 2021; Fuchs et al., 2021; Wurman et al., 2022) and the RL trajectory tracker, with their underlying architecture, environment interaction, and hyperparameters. The used environment is based on an adapted version of the F1TENTH gym racing environment (O’Kelly et al., 2020b). Both RL agents were implemented using the Stable Baselines 3 (SB3) Soft Actor Critic (SAC) algorithm (Haarnoja et al., 2018), 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 \(\gamma\)
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. Reinforcement learning 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.

at 0.99, an episode length of 10000, batch size of 64, train frequency of 1, and using the Multilayer Perceptron (MLP) policy.

2.2.1. End-to-End Racer
To generate a baseline for comparisons, we utilized the frequently used model-free end-to-end architecture of Chisari et al. (2021), Song et al. (2021), and Fuchs et al. (2021). 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 Chisari et al. (2021), Song et al. (2021), and Fuchs et al. (2021): $\text{obs}_{\text{Frenet}} = [p, n, \psi, v_x, v_y, \dot{\psi}] = [\text{progress along the path, perpendicular deviation from the path, relative heading, longitudinal velocity, lateral velocity, yaw rate}]$. An array of LiDAR distance measurements is also included; compared to the original scan, this is downsampled by taking only one every 108th scan, making the final $\text{scan}_{\text{filtered}}$ a 10-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 Figure 3, as well as a diagram of the Neural Network (NN) in Figure 4. The agent learns a control policy with online environment interaction based on the previously defined reward function in Section 2.1. Since such advancement-based rewards were broadly tested (Brunnbauer et al., 2022; Chisari et al., 2021; Fuchs et al., 2021; Song et al., 2021), we consider this agent a reasonable comparison model.
2.2.2. **Trajectory Conditioned Driver**

The proposed trajectory tracker TC-Driver tracks the spatial trajectory generated by a high-level planner. Within this work, a pre-generated Model Predictive Contouring Controller (MPCC) trajectory is used for training, which has been custom implemented for this task, following Liniger et al. (2014). 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, for example by using the centerline 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 \( obs_{traj} = [\text{relative trajectory, scans, progress along the path, perpendicular deviation from the path, relative heading, longitudinal velocity, lateral velocity, yaw rate}] \). 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 schematic overview of the RL environment interaction with the TC-Driver agent is visible in Figure 3, as well as a diagram of the NN in Figure 4. The reward function is as defined in Section 2.1.

2.3. **Tire Parameter Randomization**

The F1TENTH simulation environment utilizes the single-track dynamic model of Althoff et al. (2017). 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 \( \mu_{\text{noisy}} \sim N(1.0489, 0.0375) \).

3. **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,
TC-Driver: A trajectory conditioned reinforcement learning approach to zero-shot autonomous racing

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. Left is the end-to-end agent; middle shows the proposed TC-Driver; right shows MPC. Crosses are used to indicate crashes into the track walls, as it can be seen, only TC-Driver manages to drive in the shown chicane. Models were tested for 200 runs on the training track $F$.

3.1. Robustness to Tire Modeling Mismatch

To test the capabilities of the algorithms to generalize to different tire friction, 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 Figure 5 one can see a trajectory extract, with the 200 laps superimposed one on the other.

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 2.5%, 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 to 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 to 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 Jain et al. (2021). 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.
3.2. Track Generalization Capabilities

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 Figure 2. The tracks were obtained from the open-source repository accompanying Heilmeier et al. (2020). 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 Heilmeier et al. (2020), instead of the MPCC traversed trajectory as utilized during training.

Table 3 depicts the described runs on the unseen tracks visible in Figure 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, 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, we notice that 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 lengths, 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 Figure 6, as a part of the track *Oschersleben*. Here we can see two of the high-speed chicanes, and we can see that TC-Driver also occasionally fails at driving in this situation.

### 3.3. Computational Time

We focus on the computational time of the utilized control methods. Table 4 depicts the average computational 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.

### 3.4. Sim2Real Capacities

To investigate and validate the simulation results of the proposed architecture, the Sim2Real capability of TC-Driver is demonstrated on a physical race car in the 1:10 form factor, namely, the F1TENTH platform (O’Kelly et al., 2020a). The robot is built upon the off-the-shelf *Traxxas 4x4 Slash* race chassis and power train, driven by a *VESC 6 MkIV* Electronic Speed Controller (ESC). Furthermore, the robot sensors consist of the integrated inertial measurement unit of the ESC, as well as a *Hokuyo UST-10LX* laser range measurement sensor. The onboard computer is a *NUC10i5FNKNI3-10110U* running standard Ubuntu 20.04 and Robot Operating System (ROS) *Noetic*. A state estimator as well as 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 algorithm based on Hess et al. (2016) for positional estimates and an extended Kalman filter based on Moore and Stouch (2014) for velocity estimates, ultimately allowing for the estimation of the complete dynamic state as in Polack et al. (2017) and Althoff et al. (2017). The trajectory planner computes a globally optimal trajectory of a given track, with respect to minimum curvature, based on Heilmeier et al. (2020), which is the same planner as used previously in Section 3.2. Finally, a 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 racetrack setup are shown in Figure 7.

#### 3.4.1. 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 10 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 3. They also have the same action space and run on the same physical platform on the same track, thus serving fair comparison conditions.
As can be seen in Figure 8, the proposed TC-Driver yields superior zero-shot generalization capabilities when compared to 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, 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. Interesting to mention is that TC-Driver manages to retain similar metrics in terms of lap time standard deviation $t_{\sigma}$ and crash ratio, as achieved in the simulation 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.

4. Conclusion

This paper presented TC-Driver, a hybrid RL approach to autonomous racing, that utilizes 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 or 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 (O’Kelly et al., 2020b) as well as on the physical F1TENTH platform (O’Kelly et al., 2020a). The proposed TC-Driver architecture shows that it can adapt to model mismatch scenarios that a non-learning-based MPC fails to handle (Liniger, 2021). It achieves lower and more consistent lap times, compared to the end-to-end agent based on Chisari et al. (2021), Fuchs et al. (2021), and Song et al. (2021), and has by far the lowest overall crash ratio in the model mismatch setting (MPC, 80.50%; end-to-end, 73.50%; 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%; TC-Driver, 8%). Lastly, 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 $t_\sigma \sim 0.373$ s and displays a 10-fold 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 Chisari et al. (2021) or the introduction of Bernoulli policies (Seyde et al., 2021). Lastly, a highly interesting RL approach would be the utilization of a model-based RL architecture as well as the integration of a recurrent neural network architecture, as inspired by Brunnbauer et al. (2022). As the computational effort required by MI 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 to 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.

Acknowledgments

We 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 steppingstone for this work.

ORCID

Edoardo Ghignone https://orcid.org/0000-0003-3843-2661
Nicolas Baumann https://orcid.org/0000-0001-6591-1321
Michele Magno https://orcid.org/0000-0003-0368-8923

References


