Hybrid Reinforcement Learning for Efficient Training of a Mobile Robot

Fedi Boukhris[1], Jens-Christian Will[1], Timo von Marcard[1], and Hanno Homann[1]

***Abstract*—Sample effective and stable training remains a key challenge in reinforcement learning (RL), especially for real-world applications such as mobile robot control where data collection is time-consuming and failures may be hazardous.**

Building on the residual reinforcement learning paradigm, this work presents, to the best of our knowledge, one of the first detailed physical studies of a residual Soft Actor-Critic (SAC) controller for camera-based lane following on a mobile robot. We combine an established stable, but sub-optimal lateral P-controller with a regularized SAC agent in a hybrid architecture. The classical controller provides baseline stability and rapid initial learning, while the RL agent learns residual corrections to improve performance. We employ a PID-inspired reward function and quadratic policy output regularization to ensure smooth control actions and effective sim-to-real transfer.

The hybrid controller design enables rapid training convergence, requiring only a few epochs and outperforming the pure RL approach by two orders of magnitude in sample efficiency. This enables efficient hyperparameter tuning in simulation and opens the door to future learning directly on physical robots. Fine-tuning with only a few dozen real-world laps achieved robust transfer to the physical robot, maintaining the same architecture and hyperparameters. The method generalized effectively to new scenarios, such as lane changes.

***Index Terms*—physical reinforcement learning, hybrid architecture, mobile robot, Soft Actor-Critic, sim-to-real transfer, sample efficiency**

I. Introduction

Autonomous driving is a promising technology for improving traffic safety and efficiency. While perception has advanced significantly through machine learning, planning and control are often still based on classical model-driven approaches. These methods require extensive parameterization and struggle to adapt to complex scenarios.

RL agents learn optimal control strategies through interaction with the environment, without explicit physical models [1]. Deep Reinforcement Learning (DRL) combines RL with deep neural networks, allowing for the approximation of highly nonlinear functions and complex decision-making in dynamic environments [2]. DRL has proven effective for motion planning and vehicle control, particularly in stochastic and uncertain scenarios, enabling adaptive behaviors such as lane following, lane changes, and speed regulation [3]. While the majority of research addresses simulated environments, studies on physical robots remain comparably scarce [1].

[1]All authors are with the Faculty of Electrical Engineering and Information Technology, Hannover University of Applied Sciences, Hannover, Germany {fedi.boukhris, jens-christian.will, timo.marcard, hanno.homann}@hs-hannover.de

Recent surveys show that DRL for mobile navigation still suffers from poor sample efficiency and fragile sim-to-real transfer for physical robots. Residual RL and residual policy learning have been proposed to address these challenges by combining classical controllers with learned residual corrections. Initial work focused on manipulation tasks [4], [5], demonstrating that RL agents can learn corrective actions on top of stable baseline controllers. This paradigm has been extended to residual reactive navigation for obstacle avoidance [6], and hybrid path-following approaches combining classical steering with DRL velocity control [7]. However, existing residual RL and hybrid navigation methods do not systematically address sample-efficient training and sim-to-real transfer for camera-based lane following with residual lateral control. Specifically, prior work does not target lane/path following with image-based perception and residual lateral control.

This paper contributes to bridge this gap by providing an applied study and analysis of residual RL for mobile robot lane following with camera input, focusing on sample efficiency and sim-to-real transfer. We report two-orders-of-magnitude training speedups with quantitative sample-efficiency analysis and analyze PID-structured reward plus action regularization. To isolate the effect of residual learning, a proportional lateral controller is intentionally selected as a simple and transparent baseline. More advanced control strategies, including PID, Pure Pursuit, and MPC, may later be integrated into the hybrid framework.

II. Related Work

A. Classical Control for Mobile Robots

Classical control techniques such as PID controllers, adaptive control, and Model Predictive Control (MPC) have been widely used for lateral and longitudinal vehicle control.

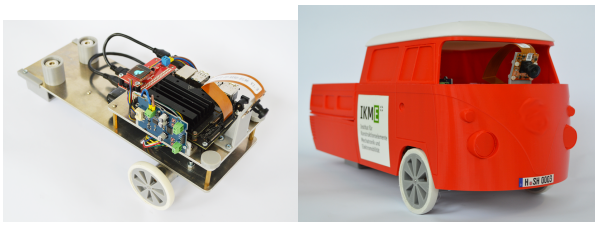

Fig. 1. Robot hardware with front-facing camera, controller board visible (left) and with body shell installed (right)

While these methods are reliable in structured environments, they often struggle with nonlinear dynamics, unexpected disturbances, and complex scenarios [8]–[10]. MPC offers predictive optimization but is more sensitive to the accuracy of the underlying model. These limitations motivate the use of learning-based approaches that can adapt to complex, dynamic environments.

B. DRL for Path Following and Navigation

Recent DRL-based control architectures vary in their use of sensor modalities and state representations. End-to-end approaches, such as DeepRacer [11], use raw camera images to generate steering commands. Other works employ neural encoders or autoencoders to extract compact latent states from sensor data [12], [13]. Multi-modal systems combine camera and LIDAR data for enhanced perception [14]. Structured approaches reconstruct geometric features, such as lateral distance and heading angle, from camera images and use them as input for RL agents [15].

Various DRL algorithms have been applied to vehicle control. Deep Q-Networks (DQN) rate the quality of discrete actions. To handle continuous action spaces typical in vehicle control, Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) have been employed for simultaneous lateral and longitudinal control [16]. Soft Actor-Critic (SAC) builds on TD3 by using a stochastic policy with entropy maximization to improve exploration and prevent premature convergence [17], [18]. SAC is a modern, state-of-the-art algorithm well-suited for continuous action spaces: its entropy-regularized objective promotes robust exploration while its off-policy nature allows efficient replay-buffer reuse, reducing the number of required environment interactions. Compared to TD3 and DDPG, SAC's stochastic policy is more resilient to reward scaling and local optima. Compared to PPO, its off-policy training is substantially more sample-efficient, which is a critical advantage as real-world data collection is costly.

While pure DRL methods can achieve good performance, they often require long training times, typically thousands of epochs, and struggle with sim-to-real transfer. Recent work on path following [7] combines Pure Pursuit for lateral control with SAC for velocity control, but does not employ residual lateral control or provide detailed sample-efficiency analysis. Other DRL navigation controllers [19]–[21] show partial or no hardware validation, and limited discussion of reward/regularization design for transfer. These methods do not exploit a residual RL structure on top of a stabilizing lateral controller for fast, safe training.

C. Residual RL and Hybrid Controllers

Johannink et al. [4] pioneered residual RL for robotic manipulation, demonstrating that learned residual policies can refine classical controllers for contact-rich tasks such as block insertion. Silver et al. [5] formalized residual policy learning, showing how to combine prior policies with learned corrections. These works established the foundational paradigm but focused on manipulation tasks rather than mobile navigation.

Rana et al. [6] extended residual RL to mobile robot obstacle-avoidance navigation, using a classical reactive planner as the base controller and learning residual corrections for goal-directed navigation in cluttered indoor environments. Their approach uses LIDAR-based sensing and switches between residual and prior policies. However, this work targets obstacle avoidance rather than lane following, uses different sensing modalities (LIDAR vs. camera), does not employ PID-structured rewards for lane tracking, and does not provide detailed sample-efficiency studies.

Gao et al. [21] proposed a hybrid approach combining given control with DDPG for tracking control of nonholonomic robots. However, their work is simulation-only and does not address camera-based lane following or employ residual SAC with systematic sim-to-real validation.

In contrast to prior residual RL work in manipulation [4], [5] and obstacle-avoidance navigation [6], this paper targets camera-based lane following with a residual SAC lateral controller. We focus on systematic evaluation of sample efficiency (achieving $\sim$100$\times$ faster convergence than pure RL) and employ a PID-inspired reward combined with action regularization that enables efficient sim-to-real fine-tuning on a physical mobile robot.

D. Sim-to-Real Transfer for Mobile DRL

Sim-to-real transfer remains a critical challenge for deploying DRL on physical robots. Various approaches have been proposed to bridge the simulation-reality gap, including domain randomization, expert demonstrations, and reality-sim-reality (RSR) loops [19], [22]. Policy output regularization has proven particularly effective by penalizing large action changes to ensure smooth control [23]. In this work, we demonstrate that combining PID-inspired reward design with quadratic action regularization within a residual architecture enables robust sim-to-real transfer for camera-based lane following, requiring only a few dozen real-world training laps for fine-tuning.

Conceptually, our method follows the residual RL paradigm [4], [5]. Our main contributions are:

- Residual RL for mobile lane following: An applied instantiation of residual RL where a lateral P-controller and a SAC residual agent jointly control a physical mobile robot for camera-based lane following and lane changes, with full sim-to-real validation.
- Sample-efficiency analysis: Quantitative comparison showing approximately two-orders-of-magnitude speedup over SAC-only baseline (convergence in $\sim$25 epochs vs. $\sim$400 epochs), enabling fast hyperparameter sweeps which is rarely reported in navigation-oriented works.
- Open and reproducible methodology: Comprehensive evaluation in simulation and real-world environments, including generalization to lane changes, with detailed architectural and hyperparameter documentation.

III. Materials and Methods

A. System Architecture

The robot platform (Figure 1) is built on a differential drive chassis and features a Jetson Orin Nano as the onboard controller, a 200° wide-angle Sony IMX219 camera for perception, and two DC gear motors with Hall sensors for controlled wheel actuation.

Our control software (Figure 2) was implemented in ROS2. In the perception module, a ResNet-18-based neural network processes camera images at 20 Hz. Geometric lane features are extracted and transformed to vehicle coordinates on the ground-plane.

The control module receives the lane features and computes the state vector from the measurements, as detailed further below. This is received by the main controller which is designed as the scope of this work. It computes the linear and rotational target velocities v and ω which are then transformed to target wheel speeds, using the differential drive kinematics.

At the actuation module, a PID wheel speed controller ensures that these target values are followed accurately, using feedback from wheel encoders (Hall sensors) at 100 Hz.

The mobile robot employs a differential drive model, allowing independent control of the left and right wheels using measurements from the wheel encoders. Since the robot can rotate in place, its translational velocity v and rotational velocity ω are decoupled and controlled independently by the motion control module.

The simulated track replicates the closed-loop course used in physical experiments, including curves and straight sections. Figure 3 compares the simulation environment and the physical track map. The simulation shows the robot's position, lateral error, heading deviation, and velocities, supporting analysis and optimization of control strategies during simulation. Only the inner lane is used for training as it's more challenging due to the narrow curves. While the training setup is intentionally kept simple, our evaluation on the physical system also addresses the outer lane as well as lane changes to investigate generalization and stability of learned control policies.

B. System State

The system state provides the observations for action selection by the RL agent and reward computation by the simulation environment. It consists of six variables chosen for precise trajectory tracking. The state vector draws inspiration from the four-component design of [18]combining lateral error and its derivatives with heading angle. We extend it with a sliding-window integral term ($N = 20$) and with linear and angular velocities, following [23] to promote smooth control and sim-to-real transfer.

$$\mathbf{s} = \begin{bmatrix} e_{lat} & \int e_{lat}\, dt & \frac{de_{lat}}{dt} & \phi & v & \omega \end{bmatrix}^\top \tag{1}$$

Figure 4 illustrates our definitions of the lateral error e_{lat} and the heading error ϕ within the robot coordinates.

The lateral error e_{lat} is computed as the lateral distance between the robot and a point on the reference trajectory at a fixed lookahead distance of 0.4 m (chosen to be within the camera's field of view on the ground surface while providing sufficient preview for control).

The integrated error captures persistent deviations. It is computed over the last $N = 20$ time steps (1 s at 20 Hz), which is long enough to compensate bias but short enough to avoid sluggish integral behavior in tight curves. This sliding-window design prevents uncontrolled accumulation over long episodes while enabling fast correction of systematic offsets.

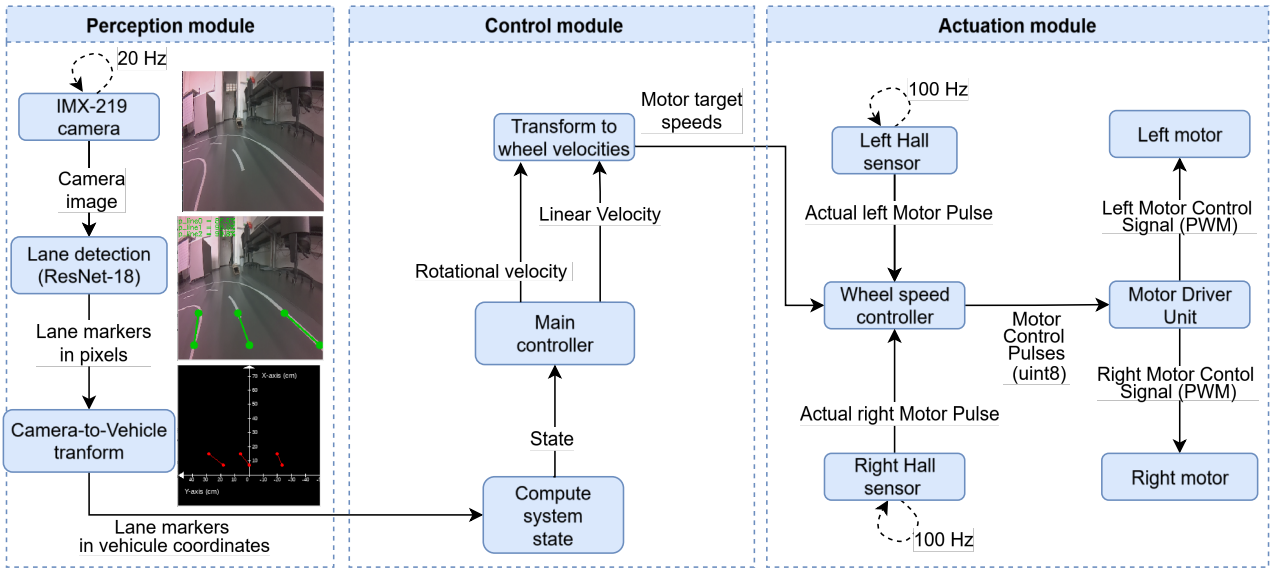

Fig. 2. Software architecture: perception, control, and actuation modules.

The rate of change of lateral error provides information about error dynamics. This enables the agent to react quickly to rapid deviations when entering curves, supporting fast corrective actions without sluggish behavior.

The orientation difference ϕ between robot and trajectory is especially important in curves. Notably, we do not integrate or differentiate ϕ to prevent overshooting in curve entry/exit. If ϕ were differentiated or integrated, rapid corrections could cause the robot to cross the lane centerline, destabilizing trajectory tracking.

Including current linear velocity v and rotational velocity ω enables the agent to regulate robot dynamics and avoid abrupt changes. Following [23], these components promote smooth actions that transfer reliably to physical hardware, improving sim-to-real robustness.

The robot is modeled as a differential-drive platform with

$$v = (v_L + v_R)/2, \qquad \omega = (v_R - v_L)/(2R), \tag{2}$$

where v_L and v_R are the wheel velocities and $2R$ is the wheelbase. With $\Delta t = 0.05\,\mathrm{s}$, the pose update is

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v_t \cos(\theta_t)\Delta t \\ y_t + v_t \sin(\theta_t)\Delta t \\ \theta_t + \omega_t \Delta t \end{bmatrix}. \tag{3}$$

Physical limits are enforced with $v_{\max} = 0.35\,\mathrm{m/s}$ and $\omega_{\max} = 3.14\,\mathrm{rad/s}$. To avoid asymmetric clipping, both wheel commands are uniformly scaled whenever one exceeds the admissible speed by

$$\beta = \min\left(1, \frac{v_{\max}}{\max(|v_L|, |v_R|)}\right) \tag{4}$$

This preserves the wheel-speed ratio and is applied consistently in simulation and real-world experiments.

C. Control Approaches

We investigated and compared three different control strategies:

a) Classical Control: As a reference, a proportional-gain controller (P-controller) was used for lateral control, computing the rotational velocity ω_{cl} from the lateral error e_{lat}:

$$\omega_{cl} = K_p \cdot e_{lat} \cdot \omega_{max} \tag{5}$$

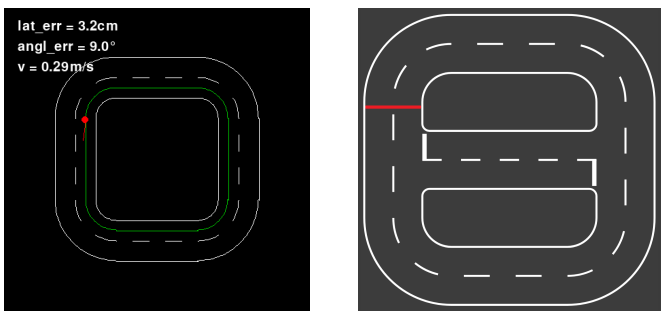

Fig. 3. Simulation environment (left) and physical track map (right).

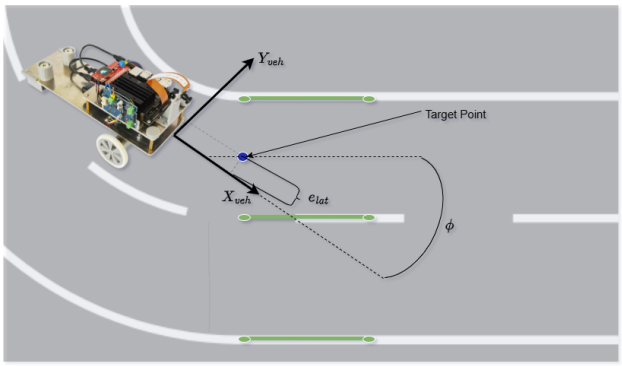

Fig. 4. Definitions of lateral error e_{lat} and heading error ϕ.

Various gain factors were evaluated, a value of $K_p = 8\,\mathrm{cm}^{-1}$ showed the most stable behavior and was used if not stated otherwise (see results section). The linear target velocity v was fixed to the maximum motor speed v_{max}. The P-controller was selected as the baseline for its simplicity, transparency, and minimal assumptions: it requires no velocity model and has a single tunable parameter, making it a reproducible and interpretable reference. Stronger baselines (PID, Pure Pursuit, MPC) are acknowledged as future work.

b) Pure Reinforcement Learning Control: The RL-only approach gives full control to the RL agent, which computes incremental changes in the linear velocity (Δv_{rl}) and the rotational velocity ($\Delta\omega_{rl}$) from the current state. Given the values from the previous time step v_{prev} and ω_{prev}, the target speeds of the motor-controller are computed as:

$$v = v_{prev} + \Delta v_{rl}, \qquad \omega = \omega_{prev} + \Delta\omega_{rl}. \tag{6}$$

c) Hybrid Control: The hybrid approach combines the classical controller and an alternative RL agent. Here, the rotational velocity is the sum of the classical controller output ω_{cl} and an RL correction $\Delta\omega_{rl}$:

$$\omega = \omega_{cl} + \Delta\omega_{rl} \tag{7}$$

The linear velocity is computed just as in the RL-only approach.

Both RL-based approaches are trained using the identical state and action definitions, the same reward function, and identical model hyper-parameters to ensure direct comparability.

D. Reward Shaping

The reward function is a central component for training the RL agent. Given the observed state, it quantitatively evaluates an action with respect to the control objectives: to minimize the lateral error e_{lat} and heading deviation ϕ, while maximizing the robot's forward velocity v.

The total reward R obtained in a single time stamp is composed of two components:

$$R = R_v + R_{lat} \tag{8}$$

Here, R_v is a velocity and heading reward:

$$R_v = \frac{v}{v_{\text{max}}} \cdot (\cos\phi - \sin|\phi|) \tag{9}$$

This term encourages forward motion while penalizing lateral deviation. The cosine term rewards trajectory alignment and the sine term penalizes sideways motion, inspired from [24]. Moreover, R_{lat} is a PID-inspired lateral reward:

$$R_{lat} = -P \cdot \left(\frac{e_{lat}}{e_{max}}\right)^2 - I \cdot \left(\frac{\int e_{lat}\, dt}{e_{max}}\right)^2 - D \cdot \left(\frac{\frac{de_{lat}}{dt}}{e_{max}}\right)^2 \tag{10}$$

This PID-inspired term penalizes instantaneous lateral error e_{lat}, its integral, and derivative, following the design of [18]. The sliding-window integral prevents uncontrolled accumulation while enabling fast corrections of systematic deviations. The derivative term allows the agent to react quickly to rapid changes. All error terms are normalized by e_{max} for consistent reward scaling.

The parameters P, I, and D in R_{lat} determine the influence of each error component and were empirically tuned for balanced learning. Practically usable values are reported in Table I.

TABLE I
REWARD FUNCTION PARAMETERS

Parameter	Value	Description
P	10	Lateral error weight
I	0.5	Integral error weight
D	0.1	Derivative error weight
e_{max}	0.05 m	Max lateral error for scaling

A training episode finished normally if the lap was completed. If the robot left the lane ($|e_{lat}| > 20\,\text{cm}$) or the simulation time limit ($t > 100\,\text{s}$) was reached, the episode was truncated. To strongly discourage such behavior, an additional penalty $R -= 100$ was then applied.

E. Agent Design

Our RL agent uses a modified Soft Actor-Critic (SAC) architecture, designed for continuous control of both translational and rotational velocity. SAC consists of an actor network and two critic (Q-function) networks, along with the two corresponding target critic networks. To reduce the overestimation bias, the minimum of the two critic estimates is used when computing target values, which stabilizes training.

The actor network receives the state vector and returns the mean and log-standard-deviation for each action dimension (Δv_{rl} and $\Delta\omega_{rl}$), using two hidden layers with 100 neurons and ReLU activation. For exploration in training, actions are sampled from a Gaussian distribution and bounded by a tanh activation to ensure physical feasibility. The critic networks estimate Q-values for a given state s_t and action a_t at each timestep t.A replay buffer stores transitions (s_t, a_t, R_t, s_{t+1}) for off-policy learning. The target networks are updated using a soft-update strategy for stability. Network updates are performed using mini-batch stochastic gradient descent.

Following [23], the SAC agent is trained using a regularized loss function:

$$L_\pi = \mathbb{E}\left[\alpha \cdot \log \pi_\phi(a_t \mid s_t) - \min(Q_{\theta_1}(s_t, a_t), Q_{\theta_2}(s_t, a_t))\right] + \mathbb{E}[\pi_\phi(s_t)]^\top M \mathbb{E}[\pi_\phi(s_t)], \tag{11}$$

where $\pi_\phi(a_t \mid s_t)$ denotes the actor network's policy and M is a regularization matrix. The regularization term $\mathbb{E}[\pi_\phi(s_t)]^\top M \mathbb{E}[\pi_\phi(s_t)]$ penalizes large action outputs, promoting smooth and gradual changes in control commands.

This quadratic policy output regularization term is known to improve sim-to-real transfer [23]. Large values of the actions Δv_{rl} and $\Delta\omega_{rl}$ are penalized to promote gradual changes of the control target values. We used a diagonal regularization matrix $M = \text{diag}(10, 10)$ for both action dimensions. The diagonal values were empirically determined to find a compromise of between smooth robot actions (i.e. to enforce small updates) and unrestricted updates (i.e. no regularization).

Table II summarizes the hyper-parameters and network architecture used for the SAC agent. The parameters are adjusted manually to achieve optimal performance and stability.

TABLE II
SAC AGENT HYPER-PARAMETERS AND NETWORK ARCHITECTURE

Hyperparameter	Value
Learning rate (lr)	0.001
Discount factor (γ)	0.99
Soft-update rate (τ)	0.005
Entropy coefficient (α)	0.02
Batch size (B)	64
Replay buffer size (N_{buffer})	10,000
Actor network	2 hidden layers, 100 neurons, ReLU
Actor input	6 (state dimension)
Actor output	2 (action dimension: Δv_{rl}, $\Delta\omega_{rl}$)
Actor output activation	tanh
Critic network	2 hidden layers, 100 neurons, ELU
Critic input	8 (state + action)
Critic output	1 (Q-value)

F. Training and Fine-Tuning

The RL agent was trained using a custom Python implementation based on PyTorch, allowing to use the same actor network in the simulation as well as in the real-world environment and to integrate the hybrid control architecture. Each training episode consisted of a complete lap, with randomized initial orientations to improve robustness. The agent interacted with the environment, selected exploratory actions, received rewards, and stored transitions in the replay buffer. Network weights were updated regularly using mini-batch samples from the buffer. After each episode, the agent was validated with exploration noise disabled, and the best-performing model parameters were saved. Training was performed on a high-performance workstation (Dell Precision 5860 Tower, Intel Xeon w7-2475X, 64 GB RAM, NVIDIA

RTX 3090), ensuring efficient data processing and rapid network updates. Roughly 300 samples were generated per episode, given lap times of about 15 s and the 20 Hz camera frame rate. Inference of the actor network was executed in real time on the robot using the Jetson Orin Nano GPU.

After training in simulations, physical fine-tuning was performed to adapt the agent's policy to real-world conditions. The hybrid RL agent was deployed on the physical robot, where it interacted with the real environment and collected new transition data. Exploration noise remained enabled to achieve randomness in the data collection. A dedicated script automated data acquisition and storage. In total, 60 real-world training runs were recorded. The collected real-world data was then used for further training (fine-tuning) of the agent's policy, to overcome the reality gap and adapt to real dynamics, disturbances, and sensor noise.

IV. Results

A. Simulation Results

Figure 5 shows representative examples of the training progress of the RL-only and hybrid agents in simulation. The RL-only agent required about 400 episodes to stabilize at lap times around 15 s, mean lateral error below 0.1 cm, and average reward reaching 0.79. Initial lap times and errors fluctuate strongly before steady improvement is reached. This indicates that the RL-only agent is able to learn an effective control strategy without the classical controller as a support.

The hybrid RL agent is initially stable due to the underlying classical controller, showing no truncated episodes. Convergence is reached significantly faster after just a few episodes. Training was repeated with several random seeds, consistently showing comparable results, demonstrating that the reported results are representative and not dependent on a particular network initialization.

Figure 6 compares the three controllers for a single simulated lap. All controllers reach the maximum linear speed. The reference controller shows larger deviations, especially in the four curves of the track. The RL-only agent and the hybrid RL agent show almost identical behavior, with considerably improved lateral and heading errors compared to the reference.

Lap times and lateral errors for all approaches, are compared in Figure 7. The classical controller serves as a reference, with performance increasing as the gain K_p increases from 2 to 11 cm^{-1}. Higher gain values were not simulated, as the physical robot is known to be unstable. Both RL-based approaches outperform the classical controller, achieving lower lap times and errors. The hybrid RL and RL-only agents reach practically identical mean lateral errors, with the RL-only agent showing a slightly lower maximum error.

Overall, the simulation results demonstrate that RL-based control enables fast, precise, and robust trajectory tracking in simulation, outperforming classical control methods in both efficiency and accuracy. The hybrid approach converges within a few epochs as it just learns an offset to the reference controller. The RL-only controller achieved comparable

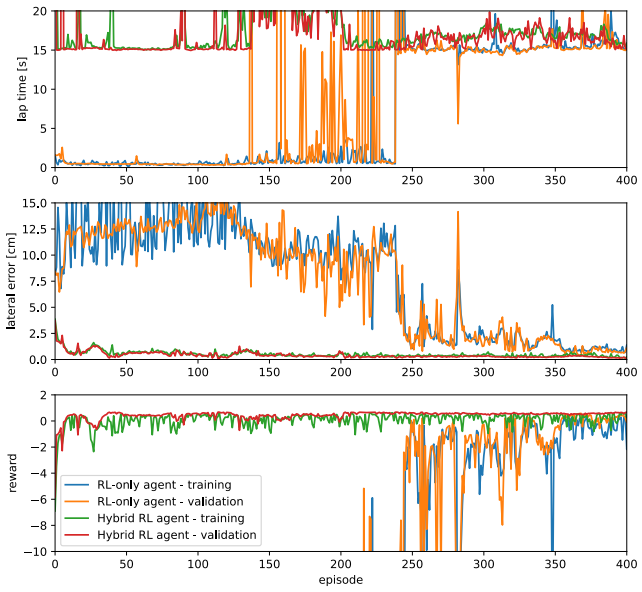

Fig. 5. Training progress of RL-only and hybrid RL control in simulation.

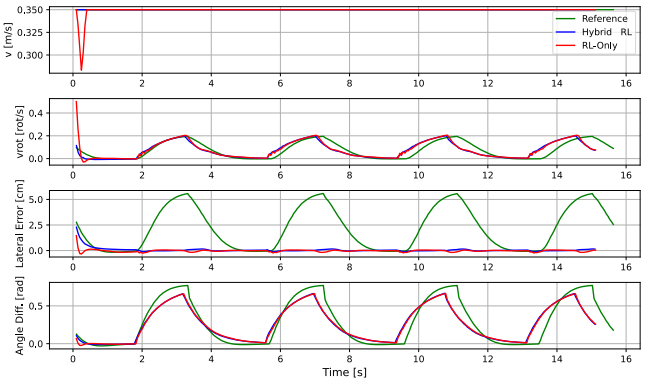

Fig. 6. Validation in simulation: comparison of reference, RL-only, and hybrid RL control for a single lap including four sharp turns.

results, showing that the actor network can fully represent the classical controller – however at the cost of significantly higher training efforts and unstable initial behaviour.

B. Real-World Experiments

To validate the developed RL approaches, experiments were conducted in the real environment using the physical robot. Both the hybrid RL approach and the RL-only approach were evaluated, alongside the classical reference controller. Additionally, a fine-tuning phase was performed to further optimize the hybrid RL agent on real-world data.

Figure 8 shows the validation results for one representative single lap in the real environment. All controllers maintained the maximum target speed of $v = 0.35\,\mathrm{m/s}$, but the RL controllers (hybrid and RL-only) achieved lower lateral errors compared to the classical reference controller.

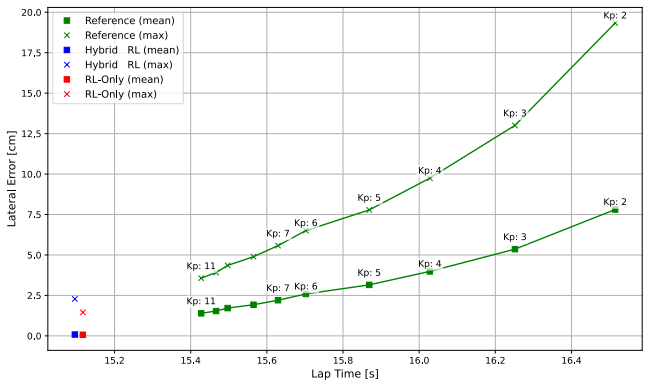

Fig. 7. Simulated lap times and lateral errors for reference, RL-only, and hybrid RL control.

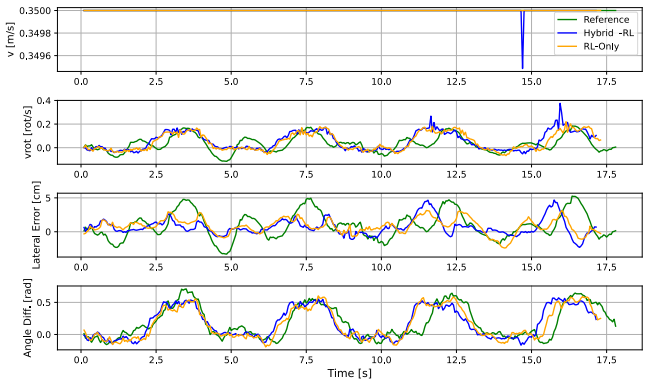

Fig. 8. Validation results in the real environment: comparison of reference, RL-only, and hybrid RL control.

Figure 9 compares lap times and errors, averaged over 10 runs, for all approaches. For the reference controller, the gain was varied from $K_p = 2\,\mathrm{cm}^{-1}$ to $11\,\mathrm{cm}^{-1}$. For small gain values up to $8\,\mathrm{cm}^{-1}$, the performance gradually improved, but for higher values the robot's rotation began to oscillate about the target trajectory, leading to increased errors and lap durations. Both RL-based controllers, and especially the fine-tuned hybrid RL agent, achieved significantly lower mean and maximum lateral errors and shorter lap times than the reference controller. The fine-tuned hybrid RL agent delivered slightly improved results in terms of accuracy and speed, compared to the hybrid RL and RL-only agents.

Table III lists the mean and maximum lateral errors and standard deviation for all approaches in both simulation and real-world tests quantitatively. The hybrid RL agent and its fine-tuned version consistently outperform the reference controller, achieving lower errors and higher stability.

C. Generalization and Lane Change

To evaluate the agent's ability to generalize, the hybrid RL agent was tested on lane change scenarios in the real environment. The lane change was initiated manually on 20 different locations along the route (including the sharp curves) at full speed. As shown in Table IV, the hybrid RL agent successfully completed all lane changes with low mean and maximum lateral errors, while the reference controller failed in all cases when driving at full speed.

Overall, the results in the physical environment demonstrate that the hybrid RL agent achieved robust, precise trajectory tracking and generalized well to new scenarios, including modified curve radius and lane changes.

TABLE III
COMPARISON OF CONTROL APPROACHES: MEAN AND MAX LATERAL ERROR IN SIMULATION AND REAL ENVIRONMENT.

Approach	Simulation			Real Environment		
	Mean [cm]	Max [cm]	Std [cm]	Mean [cm]	Max [cm]	Std [cm]
Ref. ($K_p = 8\,\mathrm{cm}^{-1}$)	1.39	3.57	1.29	1.4	5.4	1.37
RL-Only	**0.08**	**1.45**	**0.10**	1.06	3.55	0.83
Hybrid RL	**0.08**	2.28	0.20	0.97	3.90	0.85
Fine-Tuned Hybrid RL	–	–	–	**0.88**	**2.93**	**0.66**

TABLE IV
GENERALIZATION: LANE CHANGE SUCCESS RATE AND ERRORS (20 TRIALS).

Approach	Successful Lane Changes	Mean Error [cm]	Max Error [cm]
Reference	0 / 20	–	–
Hybrid RL	20 / 20	1.08	2.9

V. DISCUSSION AND CONCLUSIONS

This paper presents a hybrid control architecture combining a classical P-controller with a regularized Soft Actor-Critic (SAC) agent for efficient and robust lane following on a physical mobile robot. Geometric lane features extracted from camera images, a PID-inspired reward, and a quadratic action-regularization term together enable stable, sample-efficient training and reliable sim-to-real transfer. Our contribution is a task- and architecture-specific applied study

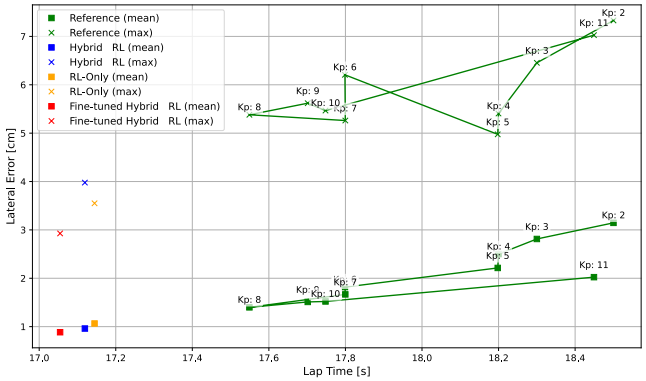

Fig. 9. Comparison of the lap times and lateral errors for reference, RL-only, hybrid RL, and fine-tuned hybrid RL control in the real environment.

of residual SAC for camera-based lane following, not a novel RL algorithm.

Both RL-based agents surpassed the P-controller reference. The RL-only agent confirmed that a SAC policy can fully represent and exceed classical lateral control, achieving a mean lateral error of 0.08 cm in simulation versus 1.39 cm for the reference. The hybrid approach required two orders of magnitude fewer training episodes than RL-only while reaching near-identical simulation accuracy, demonstrating that the P-controller provides a strong initialization that dramatically reduces the exploration burden.

Real-world experiments validated these findings on the physical robot. Both RL agents reduced mean and maximum lateral error relative to the best reference gain ($K_p = 8\,\mathrm{cm}^{-1}$), and fine-tuning on 60 real-world laps further improved accuracy to a mean error of 0.88 cm. Sim-to-real transfer required no changes to reward structure or hyperparameters. The hybrid RL agent also generalized to unseen lane-change scenarios, completing all 20 trials successfully at full speed while the reference controller failed in every case. Together, these results demonstrate that the combination of residual architecture, PID-inspired rewards, and action regularization provides a robust foundation for transfer to physical systems.

Building on these results, several promising directions emerge. Comparing residual RL against full PID, Pure Pursuit, and MPC baselines would clarify what the RL agent learns beyond classical lateral control and quantify its added value. Hardware improvements, such as higher-speed drive motors, would enable more dynamic scenarios. Evaluating the approach on more complex and diverse tracks will improve robustness against real-world challenges. Advanced hyperparameter optimization and improved lane detection could further increase trajectory-tracking precision.

All of these extensions benefit from the high training efficiency of the hybrid approach, which enables rapid adaptation to new scenarios and tasks. Ultimately, the demonstrated sample efficiency and sim-to-real robustness suggest that simulation-free training on physical robots may be feasible with appropriate safeguards.

REFERENCES

[1] Y. Chen, C. Ji, Y. Cai, T. Yan, and B. Su, "Deep reinforcement learning in autonomous car path planning and control: A survey," 2024. [Online]. Available: https://arxiv.org/pdf/2404.00340

[2] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," 2020. [Online]. Available: https://arxiv.org/abs/2001.11231

[3] F. Ye, S. Zhang, P. Wang, and C.-Y. Chan, "A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles," 2021. [Online]. Available: https://arxiv.org/abs/2105.14218

[4] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6023–6029.

[5] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

[6] K. Rana, B. Talbot, M. Milford, and N. Sünderhauf, "Residual reactive navigation: Combining classical and learned navigation strategies for deployment in unknown environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 11 493–11 499.

[7] Y. Cao, K. Ni, T. Kawaguchi, and S. Hashimoto, "Path following for autonomous mobile robots with deep reinforcement learning," *Sensors*, vol. 24, 2024.

[8] D. Li and O. Okhrin, "A platform-agnostic deep reinforcement learning framework for effective sim2real transfer towards autonomous driving," *Communications Engineering*, vol. 3, no. 1, Oct. 2024. [Online]. Available: http://dx.doi.org/10.1038/s44172-024-00292-3

[9] A. M. Annaswamy, "Adaptive control and intersections with reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 6, pp. 1–26, 2023.

[10] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 221–231, 2021.

[11] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1911.01562

[12] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1807.02371

[13] A. Pak, H. Manjunatha, D. Filev, and P. Tsiotras, "Carnet: A dynamic autoencoder for learning latent dynamics in autonomous driving tasks," 2022. [Online]. Available: https://arxiv.org/abs/2205.08712

[14] W. Hu, B. Zhao, Z. Zhou, and X. Li, "Camera-lidar fusion for deep reinforcement learning-based autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[15] N. Garnett, R. Cohen, T. Pe'er, R. Lahav, and D. Levi, "3d-lanenet: End-to-end 3d multiple lane detection," 2019. [Online]. Available: https://arxiv.org/abs/1811.10203

[16] M. Wegener, L. Koch, M. Eisenbarth, and J. Andert, "Automated eco-driving in urban scenarios using deep reinforcement learning," *Transportation Research Part C: Emerging Technologies*, vol. 126, p. 102967, 2021.

[17] H. Merton, T. Delamore, K. Stol, and H. Williams, "Deep reinforcement learning for local path following of an autonomous formula sae vehicle," 2024. [Online]. Available: https://arxiv.org/abs/2401.02903

[18] H. Wang, L. Feng, Y. Zhang, J. Zhou, and H. Du, "Human-machine authority allocation in indirect cooperative shared steering control with td3 reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 6, pp. 7576–7588, 2024.

[19] R. Chai, H. Niu, J. Carrasco, F. Arvin, H. Yin, and B. Lennox, "Design and experimental validation of deep reinforcement learning-based fast trajectory planning and control for mobile robot in unknown environment," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, pp. 5778–5792, 2022.

[20] Y. Ou, Y. Cai, Y. Sun, and T. Qin, "Autonomous navigation by mobile robot with sensor fusion based on deep reinforcement learning," *Sensors*, vol. 24, 2024.

[21] X. Gao, R. Gao, P. Liang, Q. Zhang, R. Deng, and W. Zhu, "A hybrid tracking control strategy for nonholonomic wheeled mobile robot incorporating deep reinforcement learning approach," *IEEE Access*, vol. 9, pp. 15 592–15 602, 2021.

[22] N. Liu, Y. Cai, T. Lu, R. Wang, and S. Wang, "Real-sim-real transfer for real-world robot control policy learning with deep reinforcement learning," *Applied Sciences*, vol. 10, 2020.

[23] E. Chisari, A. Liniger, A. Rupenyan, L. van Gool, and J. Lygeros, "Learning from simulation, racing in reality," 11/26/2020. [Online]. Available: http://arxiv.org/pdf/2011.13332v2

[24] B. Lau. (2017) Using keras and deep deterministic policy gradient to play torcs. [Online]. Available: https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html