Online Dynamics Model Learning and Policy Fine-tuning for Autonomous Racing

Ethan Fan
University of Pennsylvania
Philadelphia, United States
ethanfan@seas.upenn.edu

Shriyash Bajaj
University of Pennsylvania
Philadelphia, United States
bajaj10@seas.upenn.edu

Zixuan Bian
University of Pennsylvania
Philadelphia, United States
bianzx@seas.upenn.edu

Thomas Stephen Felix
University of Pennsylvania
Philadelphia, United States
stfelix@seas.upenn.edu

***Abstract*—Reinforcement learning (RL) policies trained in simulation for high-speed autonomous racing suffer from a compounding sim-to-real gap: physics engines model tire slip, servo lag, and chassis flex inaccurately, and these errors grow nonlinearly with vehicle speed. We present an online adaptation framework for the F1Tenth platform that iteratively closes this gap without requiring ground-truth actuator models or additional sensor hardware. A base policy is trained offline using Proximal Policy Optimization (PPO) with domain randomisation in a LiDAR-based simulation environment and deployed via ONNX to the onboard Jetson computer. During each on-track session, a dynamics network is trained from scratch on real IMU and odometry data, learning to predict yaw rate and lateral velocity from a 15-step history of drive commands and vehicle states. This learned model is injected into the simulator, the policy is fine-tuned with PPO against the corrected dynamics, and updated weights are hot-swapped onto the car without interrupting operation. Experiments on the Levine Hall second-floor corridor track demonstrate that each adaptation cycle produces a policy better suited to the specific car and track conditions, enabling progressively higher operating speeds with maintained stability.**

***Index Terms*—sim-to-real transfer, learned dynamics, reinforcement learning, autonomous racing, F1Tenth, domain adaptation, online learning, PPO**

I. Introduction

Simulation-to-real (sim-to-real) transfer is one of the central challenges in applying deep reinforcement learning to physical robotic systems. Policies trained in simulation routinely fail or degrade on real hardware because no simulator perfectly captures the full complexity of real-world dynamics [5]. For high-speed ground vehicles this problem is particularly acute: at low speeds a rigid-body kinematic bicycle model is an adequate approximation, but as speed increases tire slip, lateral load transfer, and actuator lag become dominant effects that standard simulators either ignore or model only coarsely [8]. The result is a policy that drives conservatively below the speed at which its simulation training remains valid, or that becomes unstable the moment it is pushed beyond that threshold.

The F1Tenth autonomous racing platform [1] provides a compelling testbed for studying this problem. At low speeds a policy trained in `f1tenth_gym` transfers reliably to the physical car; at racing speeds the gap between simulated and real vehicle response becomes large enough to cause oscillation, wall contact, and failure. The naïve fix—adding more domain randomisation or tuning the simulator's physical parameters by hand—is labour-intensive and does not generalise: every car has its own servo wear, floor surface, and tyre condition that are impractical to characterise offline.

We propose a different approach: *learn the gap from real driving data and close it iteratively*. Our system, illustrated in Fig. 1, operates in three stages. In the **offline base-training** stage a capable and safe policy is learned in simulation using PPO [2] with domain randomisation [3]. In the **dynamics identification** stage the car drives laps at a moderate speed and an onboard data collector records steering commands, yaw rate, and lateral velocity from the IMU and wheel odometry. A neural dynamics network is then trained on this data to predict real vehicle response from the command-and-state history, capturing lag, slip, and nonlinearity end-to-end without decomposing them into named physical sub-models. In the **adaptive policy refinement** stage this learned dynamics model is injected into the simulator and the policy is fine-tuned with PPO against the corrected environment. The updated ONNX model is then hot-swapped onto the car for the next session, completing one cycle of the adaptation loop.

The core contributions of this work are:

- A lightweight, sensor-agnostic dynamics network that captures the full sim-to-real discrepancy of a 1:10-scale race car from IMU and odometry alone, requiring no servo position feedback or additional instrumentation.
- An end-to-end online adaptation pipeline—collect, retrain, redeploy— that runs on-device and requires no manual parameter tuning between cycles.
- A practical deployment study on the F1Tenth platform documenting the engineering challenges of matching simulation and real sensor cadences, selecting a geometrically consistent test environment, and achieving reliable hot-swap weight updates.

II. Related Work

A. Sim-to-Real Transfer and Domain Randomisation

Sim-to-real transfer has been extensively studied in manipulation and locomotion. Tobin *et al.* [3] showed that randomising visual appearance in simulation is sufficient for robust real-world object pose estimation, while Peng *et al.* [4] demonstrated that randomising physical dynamics parameters—mass, friction, and actuator delay—enables legged locomotion policies to transfer without further adaptation. Our work employs domain randomisation as a first step but treats it as insufficient at high speeds, complementing it with an online learned correction.

B. Learned Dynamics Models

Rather than randomising fixed simulator parameters, an alternative is to learn a residual or full dynamics model from real data and use it during policy optimisation. Nagabandi *et al.* [6] showed that a neural network trained on transition data can serve as an environment model for model-based RL, with model-free fine-tuning used to correct for compounding model errors. Hewing *et al.* [7] survey the broader class of learning-based model predictive control methods, including Gaussian process and neural residual models. In the context of autonomous racing specifically, Kabzan *et al.* [8] demonstrate that a Gaussian process residual model learned from on-track data substantially reduces lap times on a full-scale autonomous race car. Our dynamics network follows the same data-driven philosophy but targets the F1Tenth scale, uses a pure neural architecture, and is embedded directly in the RL fine-tuning loop rather than the MPC planning loop.

C. Autonomous Racing with Reinforcement Learning

Model-free RL has been applied to autonomous racing at multiple scales. Chisari *et al.* [9] train a policy in the CARLA simulator and show that targeted domain randomisation narrows the sim-to-real gap for camera-based racing. Bosello *et al.* [10] specifically study F1Tenth, finding that domain randomisation is necessary but not always sufficient for high-speed transfer and advocating for additional online adaptation. Our work extends this line by pairing the randomised base policy with a learned dynamics correction, yielding a more principled and automated adaptation mechanism.

D. Online and Iterative Policy Adaptation

DAgger [11] established the principle that iterative on-policy data collection improves imitation learning. In RL the analogous idea is that policies should be retrained on data collected under their own induced distribution; our adaptation loop realises this by retraining both the dynamics model and the policy on the most recently collected on-car trajectories before each new deployment. Mehta *et al.* [12] show that actively selecting which dynamics parameters to randomise accelerates sim-to-real convergence; a natural extension of our work would apply similar active selection to the dynamics network training distribution.

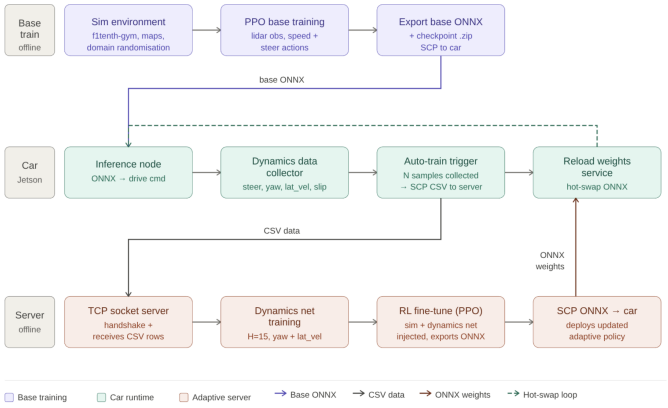

Fig. 1. End-to-end adaptive training pipeline. The base policy (blue, offline) is trained in simulation and deployed to the car via ONNX. During on-track operation (green, car runtime) a dynamics data collector records real vehicle response, which triggers an auto-train cycle on the server (red). Updated ONNX weights are hot-swapped onto the car to close the loop (dashed green arrow).

III. Methodology

A. Base Policy Training

We train a base policy in `f1tenth_gym` using PPO with the `sim2real_e2e.yaml` configuration. The simulator runs at $\Delta t = 0.01$ s on the Levine Race2 map with 1080 raw LiDAR beams, which are downsampled to 108 beams and clipped to 10 m. The deployed observation type is `lidar_state`: 108 normalized LiDAR ranges plus normalized forward velocity, yaw rate, and previous action (112 dimensions total). The action space is continuous with two outputs in $[-1, 1]$, decoded to steering (± 0.4189 rad) and speed command in $[0.5, 3.0]$ m/s.

The reward uses cross-track and heading terms (`cth`) with additional collision penalty, wall-proximity penalty, steering-change penalty, and small survival reward to encourage smooth, non-colliding laps. To improve transfer, we apply curriculum domain randomization over friction, mass, LiDAR noise/dropout, and action delay (up to 3 simulation steps). PPO is trained with learning rate 3×10^{-4}, rollout length 2048, batch size 256, 10 epochs per update, $\gamma = 0.99$, $\lambda_{\text{GAE}} = 0.95$, clip range 0.2, and entropy coefficient 0.001.

B. Dynamics Network

After deployment, we fit a lightweight actuator dynamics model from real driving logs. The network (`ActuatorNet`) is a feedforward MLP (`[64, 64, 32]` hidden units with ELU activations) trained to predict next-step yaw rate. Inputs are a 3-step history of {commanded steering angle, measured yaw rate, measured forward speed}, flattened to a 9-dimensional feature vector. Training data are standardized with `StandardScaler`; target yaw rate is standardized independently.

Supervised training uses MSE loss, Adam optimizer (10^{-3}), cosine-annealed learning rate, 300 epochs, and a 90/10

train/validation split. The best validation checkpoint is exported as TorchScript (`actuator_net.pth`) along with input/output scalers (`scaler_X.pkl`, `scaler_y.pkl`).

C. Adaptive Policy Fine-tuning

The learned actuator model is injected into RL training through the environment wrapper. During fine-tuning, the training server sets environment variables (`ACTUATOR_NET_PATH`, scaler paths, and history length) so the simulator uses the learned actuator response during policy rollouts. PPO fine-tuning is run for short online iterations (3000 steps, typically 12 parallel environments) and resumes from the previous adaptive run directory when available, otherwise from the base checkpoint.

After each fine-tune iteration, the new policy is exported from SB3 to ONNX via `export_model.py`. The server then transfers the ONNX file to the car over SCP and returns the remote path. The collector calls the car-side `/reload_weights` service, and the inference node atomically swaps to the new ONNX session without restarting ROS nodes, enabling iterative adaptation between driving sessions.

D. Data Collection and Triggering

A ROS2 collector node subscribes to `/drive`, `/odom`, and `/scan`, logging synchronized rows containing commanded steering/speed, measured speed and yaw rate, pose deltas, and active curriculum limits. Data are buffered and flushed to CSV locally. Auto-training is trigger-based: the first request is sent after `auto_train_initial_points` (configured as 5000 in our pipeline), then every `auto_train_interval_points` (10000) thereafter, with a minimum-sample guard (`auto_train_min_samples`=2000).

Car-to-server communication uses a TCP socket protocol with two phases: (1) `hello` handshake (session ID plus base model transfer on first contact), and (2) `train` request with CSV payload. The server maintains per-session state (fine-tune count and resume source), trains ActuatorNet, fine-tunes PPO, SCPs updated ONNX weights back to the car, and returns runtime parameters (max speed and safety margin) for the next deployment cycle.

IV. Results

V. Challenges

Deploying the full adaptive pipeline on physical hardware surfaced several engineering challenges that are not apparent in purely simulated evaluations.

Lidar frequency mismatch. The `f1tenth_gym` simulation ran at a higher update rate than the real Hokuyo LiDAR sensor, causing the base policy to observe scan data more frequently in training than at test time. This discrepancy produced jittery steering commands on the car. The issue was resolved by matching the simulator's step frequency to the real sensor cadence, ensuring that the policy's temporal expectations were consistent across environments.

Scan inconsistency across runs. Early experiments conducted in corridors with moving people and reflective glass produced highly variable LiDAR returns, making the policy's observations non-stationary and its behaviour unpredictable across runs. To obtain a geometrically consistent test environment we selected the Levine Hall second-floor corridor, whose straight walls and absence of glass surfaces yield stable, repeatable scans that are closely reproduced by the 2-D ray-cast model in simulation.

Retraining on a new track layout. Adapting the policy to an entirely new track map requires restarting simulation training from scratch, which is time-consuming and removes the benefit of the accumulated adaptation history. We therefore fixed the track to the repository default map, on which the base policy achieves reliable operation at 1 m/s. This provides a stable foundation from which the adaptive loop can push toward higher speeds without the compounding difficulty of concurrent layout adaptation.

VI. Future Work

References

[1] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1/10: An open-source autonomous vehicles research platform," *arXiv preprint arXiv:1901.08567*, 2019.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[4] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[5] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *Proc. IEEE Symp. Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[6] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[7] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

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

[9] C. Chisari, A. Liniger, A. Censi, L. Van Gool, and E. Frazzoli, "Learning from simulation, racing in reality," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2021, pp. 8046–8052.

[10] M. Bosello, L. Tse, G. Pau, and G. Cantelmo, "Train in Austria, race in Monza: Transferability of reinforcement learning-based control in autonomous racing," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1557–1564.

[11] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 627–635.

[12] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active domain randomization," in *Proc. Conf. Robot Learning (CoRL)*, 2020, pp. 1162–1176.