PLANNING WITH LEARNING:
DESIGNING HIERARCHICAL PLANNERS WITH BC AND PPO IN F1TENTH GYM

MINGYAN ZHOU [DEREKZMY@SEAS], BIAO WANG [WANGBIAO@SEAS], TIAN TAN [TIANTAN@SEAS],

ABSTRACT. end-to-end methods with RL and IL have gained increasing popularity in the field of autonomous racing, however, they do not involve planning with horizons, nor explicit reasoning of perception-planning-control pipeline, leading strategies myopic and reactive. In this report, we introduce our hierarchical local planner developed based on BC and PPO in the F1TENTH Gym. It outputs the lateral offset values given the selected reference trajectory and performs planning for different controllers. It uses BC for path-tracking and PPO bootstrapped by BC for fixed obstacle avoidance. Our experimental results show that the planner can plan for path-tracking and fixed obstacle avoidance, and this method makes a good attempt at planning with learning-based methods.

1. INTRODUCTION

1.1. **Formulation.** The problem we want to solve is: how to design a hierarchical learning-based (Behavioral Cloning (BC) and Proximal Policy Optimization (PPO)) local planner that can be used as a developed module for different path-tracking controllers (Pure Pursuit (PP) and Model Predictive Control (MPC)) to achieve different autonomous driving tasks (path tracking and obstacle avoidance) in F1TENTH Gym simulation environment.

1.2. **Significance.** This is a valuable problem to explore because we are trying to design a learning-based local planner via RL and IL instead of solely on end-to-end approaches or rule-based robotics pipelines. We decoupled the end-to-end RL and IL methods to the workflow of "perception, planning with learning, control" workflow. This helps the ML-based strategies to have a more explicit reasoning of planning. It also provides the horizon for future planning, instead of reactive and myopic strategy. This project also enhances path-tracking tasks with obstacle avoidance.

FIGURE 1. Conceptual understanding of BC (left) and PPO (right) hierarchical planner. BC should output the trajectory points (red) to mimic the behavior of controllers navigated by offline optimized waypoints, and PPO should output lateral offsets (blue lines) to adjust original waypoints (dark red) to safe positions (red) and avoid obstacles (yellow) as a new trajectory (green) for controllers.

1.3. **Contributions.** This work makes the following contributions to the F1TENTH gym environment:

(1) Decouple end-to-end RL and IL methods (PPO and BC) into hierarchical planning modules for local planning.
(2) Implement BC to learn from path-tracking methods (PP and MPC) respectively.
(3) Deploy PPO based on BC-bootstrapped model to achieve fixed obstacle avoidance.

2. BACKGROUND AND RELATED WORKS

2.1. **The F1TENTH Platform and the F1TENTH Gym.** The F1TENTH autonomous racing platform is an open-source evaluation framework for training and validating autonomous systems. With 1/10th-scale low-cost hardware and multiple virtual environments, F1TENTH enables safe and rapid experimentation of AV algorithms in laboratory research [3]. The F1TENTH Gym is the OpenAI Gym environment for F1TENTH. The project is developed based on this simulation environment.

2.2. **Vehicle Model and Path-Tracking Controllers.** For simplicity, we use the kinematic bicycle (single-track) model for study. We introduce 2 path-tracking controllers that used in the project. First, Pure Pursuit (PP), a geometric path-tracking controller, calculates the curvature from the center rear axle to the lookahead point, therefore achieving tracking purposes [7]. Model Predictive Control (MPC) solves the optimized trajectory and outputs the desired commands via solving formalized optimization problems. By adding hierarchical planning modules like RRT* + MPC, the path-tracking methods can avoid obstacles instead of merely relying on waypoints.

2.3. **Planning Methods.** Typically, planning problems are divided into three scales: global, behavioral, and local planning. Due to factors like vehicle scale, capabilities, and the absence of traffic rules, planning on F1TENTH is often simplified to global and local planning. Here, we discuss the current major implementations case by case.

Global Planning In autonomous racing scenarios, vehicles typically follow a predefined trajectory for optimal performance, rather than a straightforward centerline. The methodology proposed in [1] obtains the optimal raceline trajectory through minimum curvature optimization.

Local Planning Several local planning methods are extensively used on the F1TENTH.

Probabilistic planning methods, RRT, RRT*, etc., can do fast planning, but the output trajectory is sub-optimal due to the randomness of sampling, undermining the planning performance. Dijkstra and A* offer promising results, however, the high computational complexity poses significant challenges in real-time applications. Due to the limitations of the Jetson Orin Nano on the F1TENTH, achieving planning even at 10Hz with a resolution of 0.1m is challenging.

For the F1TENTH platform, "Follow-The-Gap" [6] involves finding the midpoint of the maximum gap detected by lidar. However, this approach lacks future planning due to the ignorance of the horizon. Moreover, relying solely on lidar scans can result in instability of driving due to reaction and noise.

Besides, there are some trials with learning-based methods on local planning. One good example is introduced in [9], where CNN and RNN are deployed to output the local trajectories based on inputted images. Inspired by this method, a similar hierarchical design is adopted in our project.

2.4. **Reinforcement Learning.** In a nutshell, Reinforcement Learning (RL) is the study of agents and how they learn by trial and error. Rewarding or punishing an agent for its behavior will make the agent more likely to repeat or forego that behavior in the future [4]. We introduce RL in the following several aspects.

Proximal Policy Optimization Proximal Policy Optimization (PPO) is a popular policy gradient method for RL developed by OpenAI. It aims to improve upon previous policy gradient methods by reducing the likelihood of performance collapse during training, which balances the trade-offs between simplicity, sample efficiency, and ease of tuning. The core part of PPO is the clipped objective function [5] that modifies the policy gradient update to constrain the policy update step, ensuring that the new policy is not too different from the old policy.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{1}$$

where 2 is the probability ratio of action a_t under the new policy π_θ to the old policy $\pi_{\theta_{old}}$. $\hat{A}_t$ is an estimator of the advantage at time t and ϵ is a small constant between 0.1 and 0.2. This clipping mechanism encourages modest updates to the policy, reducing the risk of making detrimental updates that could lead to performance degradation. Due to robustness and effectiveness across diverse environments, PPO has been widely adopted in various domains.

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \tag{2}$$

CleanRL CleanRL [2] is a Deep RL (DRL) library that provides single-file implementations of online DRL algorithms. Known for its simplicity and ease of research, CleanRL includes features such as benchmark implementations, tensorboard logs, local reproducibility, etc. CleanRL is designed to be used directly from its codebase, rather than as a modular library, making it well-suited for detailed research and rapid prototyping in DRL research.

F1TENTH RL There is some great development of RL on the F1TENTH platform. The implementation of PPO discussed in [11] provides an end-to-end solution for RL-based navigation, utilizing lidar scans as input to generate steering and speed commands for driving. However, this method may have several disadvantages. First, end-to-end methods fail to express logic reasoning explicitly. Besides, solely relying on lidar scans leaves the system susceptible to noise and reflections, diminishing performance into wobbly movement in real-world scenarios. Moreover, the absence of a horizon results in a reactive, myopic strategy devoid of proactive planning, further limiting its effectiveness.

2.5. **Imitation Learning.** Imitation Learning (IL) is a promising method since given demonstrations, IL trains a policy to mimic demonstrations [10], which is effective and straightforward. IL covers Behavioral Cloning (BC), Direct Policy Learning (DPL), and Inverse Reinforcement Learning (IRL). We extend the discussion of BC following the notation and explanation of [10], where it can be formulated as (3):

$$\underset{\theta}{\operatorname{argmin}} E_{(s,a^*)\sim P^*} L(a^*, \pi_\theta(s)), \tag{3}$$

specifically, $P^* = P(s|\pi^*)$ defines the distribution of the states visited by expert. We assume the perfect imitation so far and learn to continue imitating perfectly. Therefore, we minimize the 1-step deviation error along the expert trajectories. Note that the distribution of the training data is provided exogenously. By querying an expert, it provides demonstrations, treats the demonstration as an IID state-action pair, and trains using SVL.

Plenty of IL methods including BC have been implemented on the F1TENTH. By segmenting racetracks and then sampling expert data for every segment, BC can perform the driving that mimics the expert demonstrations of PP [8] via inputting the lidar scan and outputting the steering and speed. Still, this method has similar drawbacks we discussed in the f1tenth RL session: implicit reasoning, and myopia without future planning.

3. Approach

3.1. **System Architecture.** Following Fig. 2, we introduce the methodology and the implementation as follows:

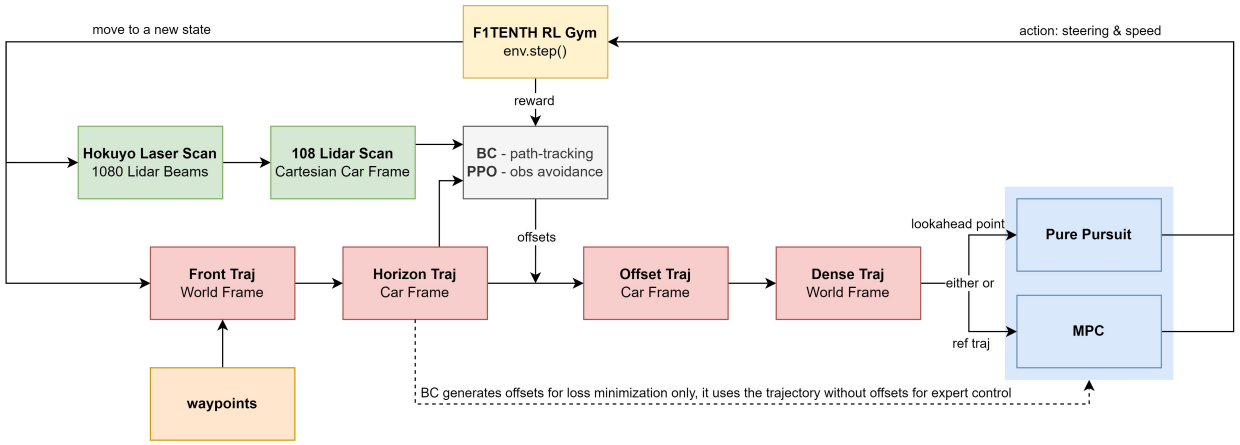


FIGURE 2. The workflow of the hierarchical planner.

Yellow - F1TENTH RL Gym Simulation A one-step simulation: given an action $a = (\delta, v)$ (steering, speed) at state $s_k = (x, y, h, v)$ (x, y positions, heading angle, and longitudinal velocity), it will return a simulated step with reward r and the next state s_{k+1} the vehicle will arrive.

Orange - Waypoints Global optimal raceline, the waypoints $\mathbf{p}$, are generated by [1] based on minimum curvature optimization. Note that waypoint data is given for this problem formulation.

Green - Perception Lidar data $\mathbf{l}^c$ downsampled and transformed from Polarized to the Cartesian car frame.

Gray - MLP / Agent The agent MLP takes the scan and the horizon trajectory and outputs the offset values $\mathbf{o}$. The offset values are the lateral offsets (blue lines in Fig. 1) corresponding to the horizon trajectory $\mathbf{t}_h^c$ (dark red points in Fig. 1) under the Frenet frame. We set the horizon $\mathrm{H} = 10$ in code. Here, we use BC to bootstrap for path-tracking, and use PPO for obstacle avoidance.

Red - Planning The process of getting horizon trajectories from $\mathbf{p}$, adding $\mathbf{o}$ onto it, and providing the densified offset trajectory to the controllers. Based on the vehicle's current pose $s^c = s_k$, the front trajectory $\mathbf{t}_f$ expresses the waypoints the car can reach in predicted time $t_{\text{pred}} = 1\text{s}$. Then, we use cubic polynomials to interpolate $\mathbf{t}_f$ and extract $\mathbf{H}$ points, and express it in car frame as horizon trajectory $\mathbf{t}_h^c$. Adding $\mathbf{o}$ onto it, we get the offset trajectory $\mathbf{t}_o^c$ (red points in Fig. 1). Finally, we transform it into the world frame and densify it to get densified offset trajectory $\mathbf{t}_d^w$. The whole planning pipeline can be expressed as a function f:

$$\mathbf{t}_d^w = f(\mathbf{p}, s_k, \mathbf{o} = \pi_\theta(\mathbf{l}^c, \mathbf{t}_h^c)) \tag{4}$$

Blue - Control 2 path-tracking controllers (PP & MPC). PP finds the lookahead point based on $\mathbf{t}_d^w$ as a new reference. We define the lookahead distance as $L = 0.8\text{m}$. Consider the simplicity of the problem, PP uses a constant speed of 2m/s, while MPC uses 2m/s as a maximum speed.

3.2. **Integrating CleanRL PPO.** Compared with end-to-end methods, implementing hierarchical planners is different from deploying PPO that directly outputs steering and speed commands. Therefore, we create a new class "F110RLEnv" to extend the features we need. Especially, in step function, instead of outputting the omnipresent "obs" data in F1TENTN Gym, we output the new observation with $(\mathbf{l}^c, \mathbf{t}_h^c, s_k)$. We also modify the logic from 2 that this function inputs offsets generated from CleanRL PPO, hence encapsulating for finishing the loop. We also provide the designed reward function penalizing the drastic offset changes, collision ($c = 1$ for collision and 0 for no collision) and rewarding the longevity for each timestep ($t = 0.01$):

$$r = 100 \cdot t - 1 \cdot ||\mathbf{o}|| - 1000 \cdot c \tag{5}$$

where we also provide several other variants of the reward function for different usages in code.

To realize the PPO functionality, we adopt ppo_continuous_action.py from CleanRL with earlier versions which is suitable for Gym for easier deployment instead of Gymnasium. For the agent MLP, we choose 4 x 256 dimensions for the actor and critic network for more non-linearity feasibility. We also add our parameters, model-saving strategies, etc. for compatibility.

Please check the project code for detailed information.

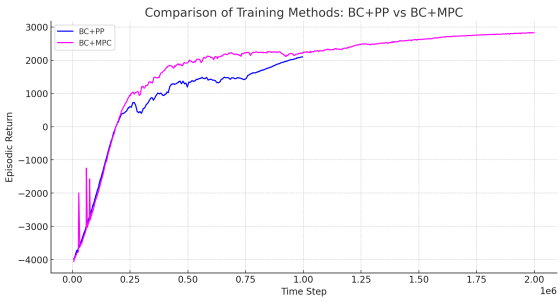
3.3. **Details in BC.** The differences between BC and PPO are that BC only uses offset values $\mathbf{o}$ for loss minimization, and the vehicle is controlled totally by controllers using waypoints. The process is denoted in the dashed line in Fig. 2. In other words, we regard PP & MPC as experts π^* to control the car for 1 lap through steps $k = 1, ..., n$. BC samples $\mathbf{o}$ in each timestep and sums every $\mathbf{o}$ in this lap. Suppose the expert controllers are perfect, hence no extra offsets are generated, then it penalizes the loss function $\mathcal{L}$ by pushing it to the trajectory generated by controllers guided by waypoints:

$$\mathcal{L} = \underset{\theta}{\operatorname{argmin}} \sum_{k=1}^{n} f(\mathbf{p}, s_k, \mathbf{o} = \pi^* = \mathbf{0}) - f(\mathbf{p}, s_k, \pi_\theta(\mathbf{l}^c, \mathbf{t}_h^c)) = \underset{\theta}{\operatorname{argmin}} \sum_{k=1}^{n} \pi_\theta(\mathbf{l}^c, \mathbf{t}_h^c) \tag{6}$$

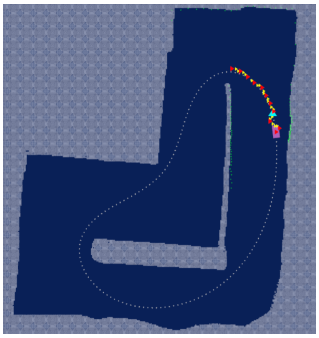
Therefore, by minimizing the sum of offsets via direct driving with PP & MPC, we can use BC to mimic the performance to learn path-tracking. This transfers the rule-based controllers into parameterized MLP planners. In code, this can be expressed easily by replacing offset trajectory with horizon trajectories in the control loop.

4. Experimental Results

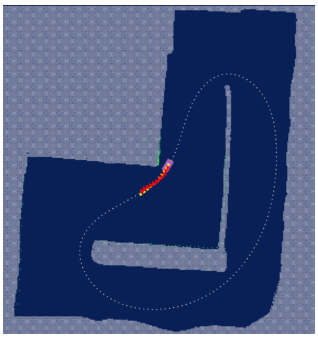
4.1. **Path-Tracking Using BC + PP & MPC.** We trained BC for path-tracking with PP and MPC respectively. Within 1 million total timesteps, BC+PP can converge to a decent performance (also for the bootstrapping for obstacle avoidance but not overfitting on path-tracking), For BC+MPC we use 2 million timesteps.



(A) Episodic returns: BC+PP (blue) & BC+MPC (magenta).



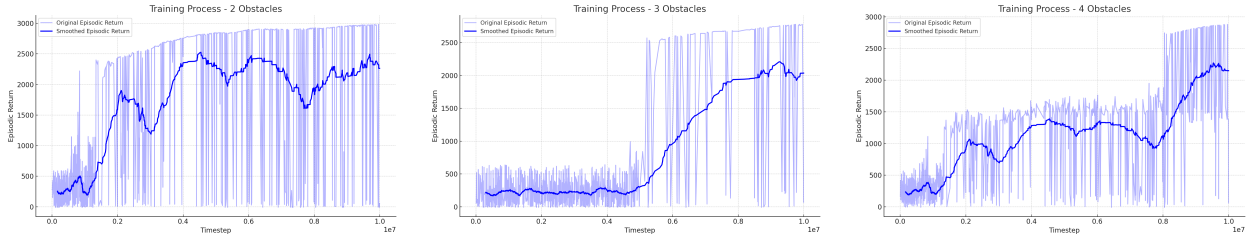
(B) BC + PP



(C) BC + MPC

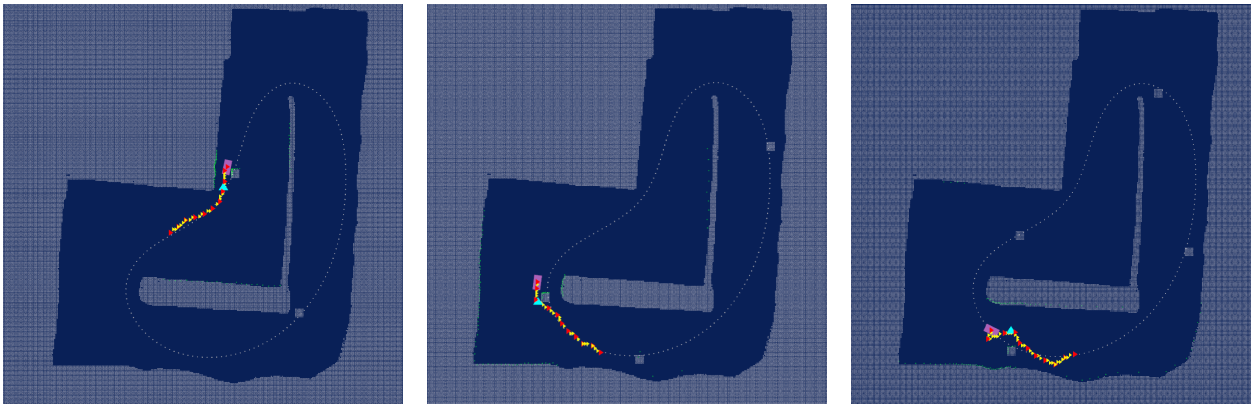
FIGURE 3. Path-tracking using BC for planning with PP & MPC. The red dots indicate the offset trajectories that the agent outputs, which productively guide the PP & MPC to track the waypoints.

4.2. **Fixed Obstacle Avoidance Using BC-Bootstrapped PPO with Pure Pursuit.** We conducted the experiments on three distinct scenarios with maps randomly populated with 2, 3, and 4 fixed obstacles with 10 million timesteps to ensure extensive training. From Fig. 5, the trained model can predict the trajectory to assist PP to avoid obstacles.



(A) Episodic returns: 2 Fixed Obstacles (B) Episodic returns: 3 Fixed Obstacles (C) Episodic returns: 4 Fixed Obstacles

FIGURE 4. Episodic returns for fixed obstacle avoidance using BC-bootstrapped PPO & PP. The returns increase through training, depicting the convergences of the policies.



(A) 2 fixed obstacles (B) 3 fixed obstacles (C) 4 fixed obstacles

FIGURE 5. Fixed Obstacle Avoidance Using BC-Bootstrapped PPO with PP. The offset trajectories (red dots and yellow lines) effectively modify the original waypoint trajectories, adjusting the lookahead point (cyan triangles) for obstacle avoidance.

5. DISCUSSION

In this final project, we successfully update the idea from end-to-end methods to using IL and RL as planners with the PP and MPC controllers. Further developments can be made by using an occupancy grid map to elaborate better planning and obstacle avoidance, boosting the generalizability by training with random static obstacles and dynamic obstacles, and decoupling the planning from decision-making to physics-constrained motion planning, etc.

REFERENCES

[1] Alexander Heilmeier et al. "Minimum curvature trajectory planning and control for an autonomous race car". In: *Vehicle System Dynamics* 58.10 (2020), pp. 1497–1527. DOI: 10.1080/00423114.2019.1631455.

[2] Shengyi Huang et al. "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms". In: *Journal of Machine Learning Research* 23.274 (2022), pp. 1–18. URL: http://jmlr.org/papers/v23/21-1342.html.

[3] Matthew O'Kelly et al. "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning". In: *Proceedings of Machine Learning Research* 123 (2020).

[4] OpenAI. *OpenAI Spinning Up - Part 1: Key Concepts in RL*. 2018. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.

[5] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[6] Volkan Sezer and Metin Gokasan. "A novel obstacle avoidance algorithm: "Follow the Gap Method"". In: *Robotics and Autonomous Systems* 60.9 (2012), pp. 1123–1134. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2012.05.021. URL: https://www.sciencedirect.com/science/article/pii/S0921889012000838.

[7] Jarrod M Snider et al. "Automatic steering methods for autonomous automobile path tracking". In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).

[8] Xiatao Sun et al. *A Benchmark Comparison of Imitation Learning-based Control Policies for Autonomous Racing*. 2022. eprint: arXiv:2209.15073.

[9] Trent Weiss and Madhur Behl. "Deepracing: Parameterized trajectories for autonomous racing". In: *arXiv preprint arXiv:2005.05178* (2020).

[10] Hoang Y. Le Yisong Yue. *Imitation Learning Tutorial*. 2018. URL: https://sites.google.com/view/icml2018-imitation-learning/.

[11] Zhijun Zhuang. *f1tenth_rl GitHub Repository*. URL: https://github.com/zzjun725/f1tenth_rl.