Investigation of Reinforcement Learning Methods for Raceline Optimization

Joseph Hadidjojo, Tung Pham

November 8, 2024

1 Introduction

Formula 1 (F1) racing is renowned for the intense precision, skill, and strategy that its drivers bring to the track. In each race, drivers must determine the fastest path around the circuit, or the "racing line," which enables them to maintain optimal speed while negotiating corners and straights. This process is not merely a physical feat; it requires an in-depth understanding of the vehicle's capabilities and a sharp sense of timing. Talented drivers often master the racing line in only a few laps of practice, fine-tuning it to suit their unique driving style and the distinct characteristics of their machinery.

Inspired by this mastery, the objective of this project is to explore the potential of reinforcement learning (RL) to approximate an optimal racing line on a simplified F1 track. The core idea is to use RL, where an agent learns through experience, to find the fastest path around a track by optimizing two key actions: acceleration and rotation. By framing the problem in this way, we aim to develop a model that could eventually serve as a foundational tool for amateur racers with limited experience.

However, this problem presents unique challenges compared to more conventional RL problems:

- **High-dimensional, continuous state and action spaces**: Unlike simpler tasks with discrete or limited action spaces, racing involves continuous control of speed, position, and rotation, all of which must be adjusted dynamically to adapt to the track layout.
- **Balancing competing objectives**: Achieving the optimal racing line requires the agent to balance maximizing speed with avoiding crashes or leaving the track boundaries. High speeds, especially around tight corners, increase the risk of the agent veering off course, demanding precise control over acceleration and rotation to maintain the ideal path within the track's limits.
- **Delayed rewards**: The consequences of an action, like setting up for a corner, may not be immediately evident. The agent must learn to associate

actions taken several turns ago with the eventual outcome, making the learning process more complex.

- **Track variability and precision demands**: Unlike simpler environments, the race track's varying curvature and the need for fine-grained control make it harder for the agent to generalize its learned strategy across different segments of the track.

These factors contribute to making the problem of finding an optimal race line particularly challenging for RL, setting it apart from simpler RL applications.

While F1 and professional racing teams likely already utilize advanced tools and systems to estimate the optimal racing line, these applications are typically proprietary and remain restricted to private use. Due to the competitive nature of the sport, insights and strategies derived from such systems are rarely disclosed to the public. Consequently, amateur racers and enthusiasts have limited access to data-driven methods for learning and improving their racing lines. By exploring this problem with reinforcement learning, we aim to create a foundation for a more accessible tool that can help individuals develop their skills and gain insights into racing line optimization without requiring the high costs or exclusivity of professional-grade systems. This project not only contributes to the field of RL in complex, continuous control tasks but also has the potential to democratize access to advanced racing analytics for a broader audience.

2 Background and Related Works

Reinforcement learning (RL) has seen significant advancements in recent years, especially in applications requiring continuous control and dynamic decision-making, such as autonomous driving and racing. In racing, RL agents must optimize not only for speed but also for safety, as they navigate complex tracks with tight turns and variable conditions. This section reviews notable works in RL and autonomous driving that have informed the development of racing agents, highlighting the techniques and challenges relevant to this project.

2.1 Deep Q-Network (DQN) for Autonomous Driving

A foundational work in deep reinforcement learning is the introduction of the Deep Q-Network (DQN) by Mnih et al. (2015) [4], which achieved human-level performance on Atari games. Although the DQN algorithm primarily focuses on discrete action spaces, it inspired adaptations for continuous and complex environments, such as autonomous driving and racing, where agents must navigate through tracks while balancing speed and control. This approach laid the groundwork for subsequent RL algorithms that could handle the continuous control demands of racing tasks.

2.2 Racing Agent with Reinforcement Learning

In racing, agents often benefit from continuous-action reinforcement learning algorithms, which allow finer control over movement variables such as speed and rotation. The Soft Actor-Critic (SAC) algorithm, introduced by Haarnoja et al. (2018) [2], is well-known for its ability to handle continuous action spaces with entropy regularization. SAC's exploration techniques are possibly quite effective for racing environments, where agents must strike a balance between fast exploration of new actions and reliable, safe driving.

2.3 Using RL for Track Navigation in Self-Driving Cars

Another significant contribution to continuous-action reinforcement learning is the Deep Deterministic Policy Gradient (DDPG) algorithm by Lillicrap et al. (2016) [3]. DDPG has shown promise in applications requiring high-precision control. By enabling continuous state and action spaces, DDPG provides a foundation for control in dynamic and high-stakes environments, which we believe makes it well-suited for racing agent applications.

By drawing on these established techniques, this project seeks to develop an RL-based racing agent capable of navigating a simplified F1 track while balancing speed and safety, making it accessible as a foundational tool for aspiring racers.

3 Technical Approach

3.1 Environment Setup

The agent operates within a racing track environment represented as a 2D continuous space. At each time step t, the agent observes the current state s_t, which includes:

- Position on the track (x, y),
- Velocity components (v_x, v_y),
- Orientation (θ).

The agent selects an action $a_t = [a_{\text{acc}}, a_{\text{rot}}]$, where:

- a_{acc} controls acceleration ($a_{\text{acc}} > 0$) or deceleration ($a_{\text{acc}} < 0$),
- a_{rot} controls rotation, with positive values turning right and negative values turning left.

The environment updates the agent's state based on these actions and the track's physics, such as track boundaries. Most real-life environmental factors such as friction, elevation, and grip condition will initially be ignored but may be considered in the future. The agent receives a reward r_t at each step, determined by:

1. Progress along the track, proportional to minimizing lap or sector times.

2. Penalties for leaving the track boundaries.

The agent learns to balance the objectives of speed and control, gradually improving its policy over training episodes.

3.2 Methodology

To optimize the agent's behavior, we leverage reinforcement learning techniques, including Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG). Each method builds upon fundamental principles of RL, such as the Bellman equation and policy optimization.

3.2.1 Q-Learning and Deep Q-Learning

Q-learning is a value-based reinforcement learning algorithm that seeks to learn the optimal action-value function $Q^*(s, a)$ using the Bellman equation:

$$Q^*(s, a) = r + \gamma \max_a Q^*(s', a),$$

where s' is the next state, and γ is the discount factor.

The algorithm iteratively updates the Q-values using [5]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q(s', a) - Q(s, a) \right],$$

where α is the learning rate. This simple, table-based approach works well in discrete environments with small state and action spaces. However, Q-learning has significant limitations when applied to complex environments like racing tracks:

- **High-dimensional state spaces**: Representing continuous states (e.g., positions, velocities, and orientations) as discrete states leads to a combinatorial explosion in the size of the Q-table.
- **Continuous action spaces**: Q-learning requires discrete actions, making it unsuitable for tasks requiring fine-grained control over continuous variables like acceleration and rotation.
- **Inefficient learning**: In large state-action spaces, learning becomes computationally expensive, and generalization across similar states is not possible.

These limitations necessitate scalable approaches like Deep Q-Networks (DQN) for handling high-dimensional state spaces. DQN extends Q-learning by approximating the Q-function $Q(s, a)$ with a deep neural network, enabling it to handle large, high-dimensional state spaces. Instead of maintaining a Q-table, DQN uses a neural network parameterized by θ to predict $Q(s, a)$ for

each possible action. This makes DQN effective for high-dimensional problems, though it remains limited to discrete action spaces.

DQN introduces two key techniques to improve stability and efficiency:

- **Experience replay**: The agent stores past transitions (s, a, r, s') in a replay buffer. During training, mini-batches of transitions are sampled randomly, reducing correlations between consecutive updates and improving data efficiency.
- **Target networks**: A separate target network $Q(s', a'; \theta^-)$ is used for calculating target Q-values. The parameters θ^- are updated periodically to reflect the weights of the current Q-network, reducing the risk of divergence during training.

To train the Q-network, DQN minimizes the temporal difference (TD) loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right],$$

where:

- θ: Represents the parameters (weights) of the current Q-network used to approximate $Q(s, a)$.
- θ^-: Represents the parameters of the target Q-network, which are periodically updated to match θ. The use of θ^- ensures stability by decoupling the target computation from the current network's rapidly changing parameters.

While DQN is effective for high-dimensional discrete action problems, it cannot handle continuous action spaces directly. To overcome this limitation, extensions such as Deep Deterministic Policy Gradient (DDPG) are used for continuous control tasks, as discussed in the next section.

3.2.2 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an actor-critic algorithm designed to handle continuous action spaces, making it particularly suitable for tasks like race line optimization, where fine-grained control over acceleration and rotation is required. DDPG builds on the Deterministic Policy Gradient (DPG) framework and incorporates elements from Deep Q-Networks (DQN) to enable stable training in high-dimensional environments.

DDPG consists of two main networks:

- **Critic Network:** The critic network approximates the action-value function, providing an estimate of the expected cumulative reward for a given state-action pair (s, a).
- **Actor Network:** The actor network outputs the optimal continuous action a for a given state s, directly mapping states to actions.

These networks are trained simultaneously, with the critic guiding the actor's policy improvement. Additionally, DDPG employs separate target networks for both the critic and actor to stabilize training where they are delayed copies of their corresponding network.

The critic network is updated using the Temporal Difference (TD) error, similar to DQN. This loss minimizes the difference between the predicted Q-value and the target Q-value, ensuring the critic learns accurate action-value estimates.

The actor network is updated by maximizing the expected Q-value as estimated by the critic. The gradient of the policy objective with respect to the actor's parameters θ_μ is [3]:

$$\nabla_{\theta_\mu} J = \mathbb{E}_s \left[\nabla_a Q(s, a|\theta_Q)\big|_{a=\mu(s)} \nabla_{\theta_\mu} \mu(s|\theta_\mu) \right]$$

This update encourages the actor to select actions that maximize the critic's evaluation, driving the policy toward higher rewards.

DDPG is well-suited for tasks requiring continuous actions because it directly outputs continuous actions, avoiding the need for action discretization. This property makes it an ideal candidate for optimizing race lines, where precise control over acceleration and rotation is crucial.

3.3 Tasks

Initially, the experiment will focus on establishing the environment and the feasibility of agent training in the environment.

The environment will be constructed with great inspiration from Gymnasium's Car Racing environment, in combination with the RocketMeister environment developed by Daniel Brummerloh [1].

Following successful construction of the environment, development of the agent implementations using DQN and DDPG will begin. If time and resources permit, further extensions may be developed and incorporated. These extensions may include, but are not limited to: adding environmental factors like tire degradation, elevation and obstacles, etc; usage of TAMER framework to introduce human reinforcement; or transforming the environment into that of a multi-agent.

4 Evaluation

The evaluation of this project will be centered on the agent's ability to optimize lap times and maintain track boundaries, with comparisons made across different reinforcement learning methods. The main evaluation criteria include lap time performance, track adherence, and generalization to multiple track layouts.

4.1 Lap Time Performance

Lap time will serve as a primary metric for evaluating each agent's efficiency in navigating the track. Rewards will be tied to lap time, either through full lap or sector times (to be determined), where a faster lap time results in higher rewards. By comparing cumulative rewards, we can measure the overall effectiveness of each method in minimizing lap time. This approach allows us to quantify each agent's speed optimization and observe improvements over training episodes.

4.2 Track Boundary Adherence

Each agent will incur penalties for going out of bounds, emphasizing the importance of maintaining control while optimizing speed. The boundary adherence metric will help determine the agent's consistency in staying within the track limits, which is crucial for real-world applicability. Frequent boundary violations or excessively aggressive maneuvers will be penalized, guiding the agent toward safer driving behaviors.

4.3 Method Comparison

To evaluate the effectiveness of various reinforcement learning algorithms, each agent will be trained and tested using different methods, DQN and DDPG (potentially other methods as well). The reward progression over time for each method will be tracked and compared, providing insights into the learning stability, convergence rates, and overall efficiency of each approach.

4.4 Track Generalization

To assess each agent's adaptability, evaluations will be conducted on three different race tracks with relatively simple layouts: Monza, Red Bull Ring, and Fuji Speedway. These tracks were chosen for their distinct characteristics, allowing us to test the agent's ability to generalize its learned racing line strategies across varied track configurations. Performance on each track will be measured independently, focusing on lap times and boundary adherence. This multi-track evaluation will help identify the robustness of each method in handling diverse racing environments.

By analyzing these metrics across different algorithms and track layouts, this evaluation will provide a comprehensive view of each method's effectiveness in race line optimization, stability, and adaptability.

5 Provisional Timeline and Individual Responsibilities

5.1 Provisional Timeline

- Early - Mid November: Environment Setup & Methods Implementation

- Mid - Late November/Early December: Methods Experimentation
- Early - Mid December: Evaluation, Report & Presentation

5.2 Responsibilities: Joseph Hadidjojo

1. Environment Setup
2. Implementing DDPG
3. Final Report

5.3 Responsibilities: Tung Pham

1. Environment Setup
2. Implementing DQN
3. Final Report

References

[1] Daniel Brummerloh. Rocketmeister. `https://github.com/danuo/rocket-meister/`, 2020.

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.