CS 5180 Reinforcement Learning

Project Report

Reinforcement Learning for Autonomous Car Racing

Girish Raut, Omkar Sargar

1. Introduction:

Our project focuses on developing an autonomous driving agent using reinforcement learning techniques, specifically designed to navigate a racing circuit. The motivation behind this endeavor is twofold: firstly, the growing interest in autonomous vehicles necessitates innovative solutions for real-time decision-making in dynamic environments. Secondly, we aim to push the boundaries of autonomous driving capabilities by venturing into the high-stakes realm of racing.

Unlike conventional autonomous driving scenarios that prioritize safety and efficiency, racing demands optimal performance and precise maneuvers. Through reinforcement learning, we strive to train an agent that can master the intricacies of racing seamlessly integrating strategies and rapid decision-making.

2. Problem Formulation:

2.1 Environment:

Initially, we explored the TORCS (The Open Racing Car Simulator) platform, which offers a comprehensive simulation environment for racing scenarios. However, we encountered challenges during the setup process, prompting us to seek an alternative solution. After careful consideration, we transitioned to the DonkeyCar Sim, a simulation platform specifically designed for autonomous driving research and development. This platform provided a user-friendly and efficient environment tailored to our needs, enabling us to focus our efforts on the core aspects of our project.

Screenshot of the experimental setup- Donkeycar Sim environment

2.1 Observation Space:

The primary sensory input to the agent is visual information captured by a front-facing camera mounted on the car. This camera provides a continuous stream of RGB images, each with a resolution of 120x160x3 pixels, offering a view of the racing environment ahead. By leveraging this visual data, the agent can perceive the intricate details of the track, obstacles, and other relevant elements, enabling informed decision-making and precise maneuvering.

2.2 Action Space:

The agent's ability to control the car's movement is governed by two continuous action spaces. Firstly, the throttle control allows the agent to regulate the engine's power output directly influencing the car's speed. Secondly, the steering output determines the car's turning direction (left or right) and the sharpness of the turn. This granular control over both acceleration and steering enables the agent to execute intricate maneuvers and optimize its performance on the racing circuit.

2.3 Reward Function Design:

The reward function is a critical component that guides the agent's behavior by providing feedback on its actions, shaping its decision-making process towards optimal performance. In our pursuit of developing a competent racing agent, we carefully experimented with different reward functions which focus on different aspects of driving. All the reward functions we experimented with incorporate the following basic components:

Speed Reward: Positive reward based on the car's velocity, encouraging faster lap times.

Collision Penalty: Negative reward for collisions.

Reverse-driving Penalty: Negative reward given for driving in the opposite direction as it is very undesirable.

3. Algorithms:

3.1 Proximal Policy Optimization (PPO):

PPO is an on-policy reinforcement learning algorithm well-suited for continuous control tasks, making it a natural fit for our racing environment. PPO aims to learn a policy that maximizes the expected reward while ensuring stability during the training process, a critical feature for complex and dynamic domains.

3.2 Soft Actor-Critic (SAC):

Soft Actor-Critic (SAC) is an off-policy algorithm that combines elements from policy gradient and Q-learning approaches, offering a unique perspective on the reinforcement learning problem. SAC aims to maximize the expected return while simultaneously considering the entropy of the policy, encouraging exploration. This exploration-driven nature of SAC can be particularly beneficial in complex environments like acing, where the agent needs to learn diverse strategies for different track sections and scenarios. By promoting exploration, SAC can potentially uncover more robust and versatile policies capable of handling the dynamic and unpredictable nature of the racing circuit.

3.3 Truncated Quantile Critics (TQC):

Truncated Quantile Critics (TQC) is a cutting-edge algorithm that builds upon the foundations of SAC, Twin Delayed Deep Deterministic Policy Gradient (TD3), and Quantile Regression Deep Q-Network (QR-DQN). Instead of predicting a single mean value for the value function, TQC employs quantile regression to predict a distribution of quantiles. It achieves this by truncating the quantiles predicted by different networks, offering a more robust and expressive representation of the value function. This approach can potentially capture the inherent uncertainty and multimodality present in complex environments like racing circuits, leading to more informed and reliable decision-making by the agent.

3.4 Comparison and Discussion:

While all three algorithms, PPO, SAC, and TQC, are well-suited for continuous control tasks and capable of tackling the complexities of the racing circuit environment, they exhibit distinct characteristics and trade-offs that warrant careful consideration.

<u>Proximal Policy Optimization (PPO):</u>

- On-policy algorithm, ensuring stable and reliable training
- Potential for suboptimal exploration in complex environments

- Less sample-efficient compared to off-policy algorithms

Soft Actor-Critic (SAC):

- Off-policy algorithm, enabling efficient use of experience replay
- Encourages exploration through entropy maximization
- Capable of learning diverse strategies for different scenarios
- Potential for overestimation of value functions

Truncated Quantile Critics (TQC):

- Captures the distribution of the value function, accounting for uncertainty
- Robust to outliers and multimodal value distributions
- Increased computational complexity due to multiple quantile predictions
- Potential for instability during training if not tuned properly

A comprehensive evaluation and comparison of these algorithms in the racing circuit environment was conducted to determine their relative strengths and weaknesses, and to identify the most suitable approach for our autonomous driving agent.

4. Experiments:

4.1 Safe Driving:

Objective: Train the agent to navigate the track safely while maximizing progress.
Action Space:

- Throttle: [0, 1]
- Steer: [-0.5, 0.5]

Reward Function:

- Penalizes going off-track, collisions and driving in reverse.
- Encourages going fast while staying safe (close to the center).

```
def calc_reward(self, done: bool) -> float:
    if done:
        return -1.0
    # Moving too far from the center
    if self.cte > self.max_cte:
        return -1.0
    # Collision
    if self.hit != "none":
        return -2.0
    # going fast close to the center of lane yeilds best reward
    if self.forward_vel > 0.0:
        return self.forward_vel
    # in reverse, reward doesn't have centering term as this can result in some exploits
    return self.forward_vel
```

4.2 Aggressive Racing:

Objective: Train the agent to adopt more aggressive racing lines while maintaining control and avoiding penalties.

Action Space:

- Throttle: [0, 1]
- Steer: [-0.5, 0.5]

Reward Function:

- Penalizes collisions and driving in reverse.
- Encourages going fast (no condition for safety).

```
def calc_reward(self, done: bool) -> float:
    if done:
        return -1.0
    # Collision
    if self.hit != "none":
        return -2.0
    # going fast
    if self.forward_vel > 0.0:
        return self.forward_vel
    # in reverse
    return self.forward_vel
```

4.3 Aggressive Racing with braking:

Objective: Train the agent to adopt more aggressive racing lines while maintaining control and avoiding penalties. Allow braking to potentially help the car to carry more speed into the corners and then brake while turning.

Action Space:

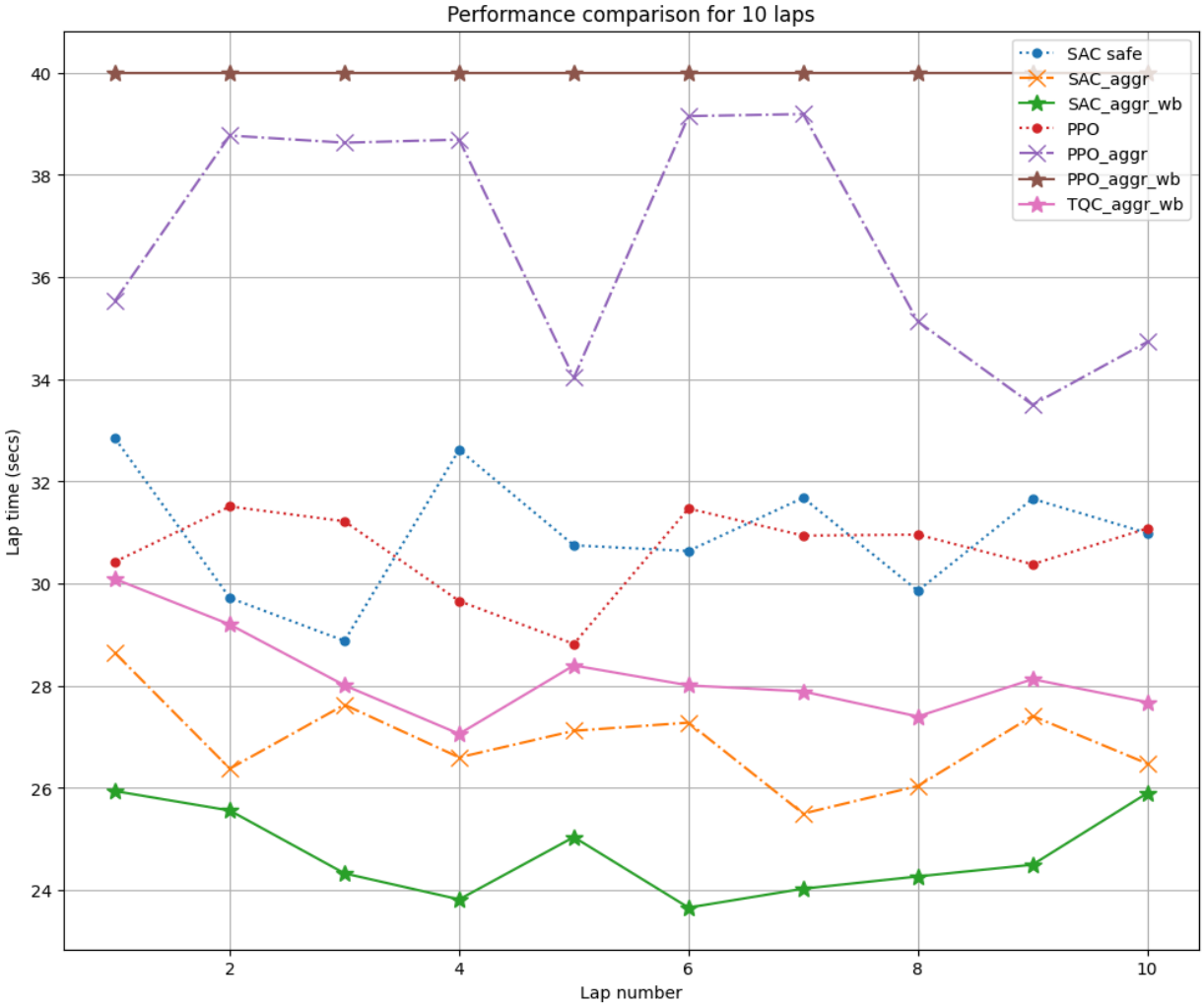
- Throttle: [-0.5, 1]
- Steer: [-0.5, 0.5]

Reward Function:

- Penalizes collisions and driving in reverse.
- Encourages going fast (no condition for safety).

```
def calc_reward(self, done: bool) -> float:
    if done:
        return -1.0
    # Collision
    if self.hit != "none":
        return -2.0
    # going fast
    if self.forward_vel > 0.0:
        return self.forward_vel
    # in reverse
    return self.forward_vel
```

5. Results

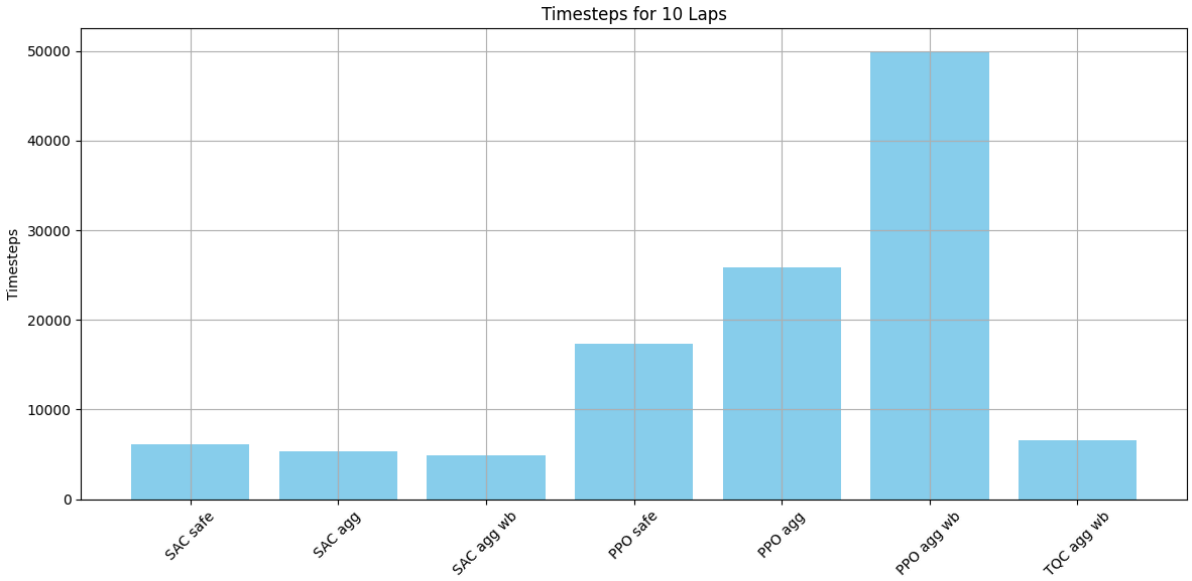


Performance as measured by lap times for all algorithms tested for 10 laps
(The data for PPO_aggr_wb is for visualization - the model was not able to complete any laps)

The models trained for aggressive driving with braking generally performed the best since it allowed the agent to take maximum advantage of the car's capabilities. When broadly comparing the different algorithms we used across all the experiments PPO's stability proved advantageous for learning baseline policies, however its on-policy nature meant that it was very sample-inefficient and was not able to discover better strategies. Conversely, SAC's off-policy approach and explicit exploration incentive facilitated the discovery of more versatile policies, albeit with the risk of overestimating value functions.

TQC, being the most advanced algorithm of the three, offers the unique ability to capture the distribution of the value function, potentially accounting for the inherent uncertainties and multimodalities present in the racing circuit. However, due to limited computational

resources we were not able to train the TQC model for as long as we trained the others. But comparing the training rewards TQC received for the ~140000 steps it was trained on with similar windows of SAC and PPO, it is clear that TQC seems to be outperforming the two.



Performance as measured by time steps taken to complete 10 laps for all algorithms
(PPO_agg_wb did not complete a single lap even after 50000 timesteps)

6. Conclusion

This project successfully developed reinforcement learning agents capable of navigating a racing circuit in a simulated environment. The agent learned to control the car's throttle and steering, achieving various objectives including safe driving, aggressive racing, and aggressive racing with braking. We explored three reinforcement learning algorithms: Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Truncated Quantile Critics (TQC).

Our analysis revealed that PPO offered a strong foundation for learning stable baseline policies. However, its data-intensive nature limited its ability to discover more advanced strategies. Conversely, SAC, with its off-policy learning and exploration incentives, excelled at finding versatile racing tactics, although it presented a potential risk of overestimating value functions.

Truncated Quantile Critics (TQC) emerged as the most theoretically promising algorithm due to its ability to capture the inherent uncertainties present in racing environments. However, limited computational resources restricted the extent of TQC's training, hindering a definitive comparison with the other algorithms. Initial results suggest that

TQC might outperform PPO and SAC, but further investigation with more training time is warranted. GitHub link.

7. Future work

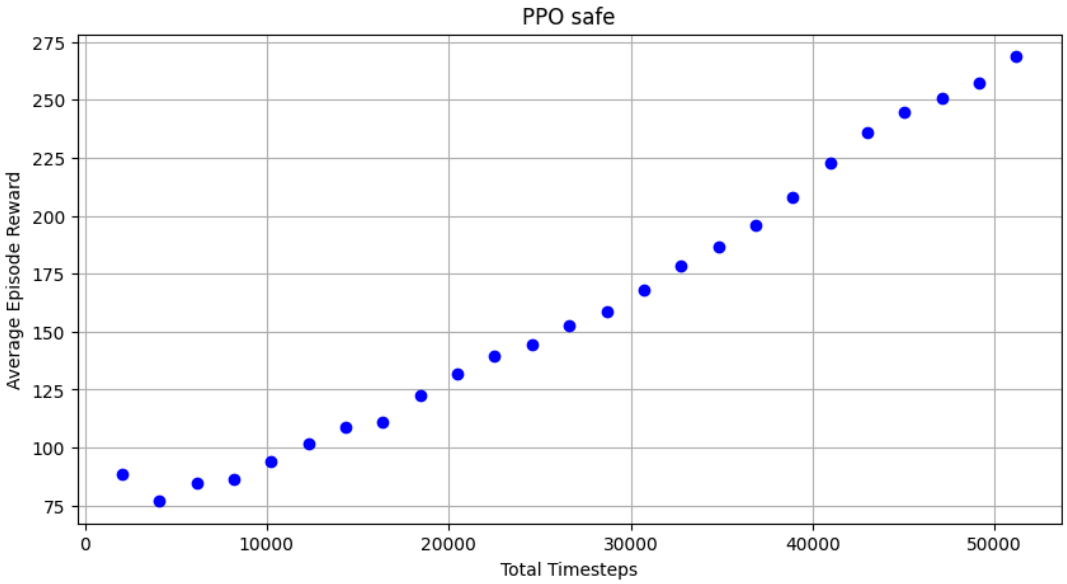
Extended Training for TQC:
Further investigate the potential of Truncated Quantile Critics (TQC) by allocating more computational resources for extended training. This will allow for a more conclusive comparison of its performance against PPO and SAC.

Advanced Reward Function Design:
Explore more sophisticated reward functions that go beyond speed and safety. These could incorporate factors like racing line optimization, cornering techniques, and car dynamics for even more nuanced control.

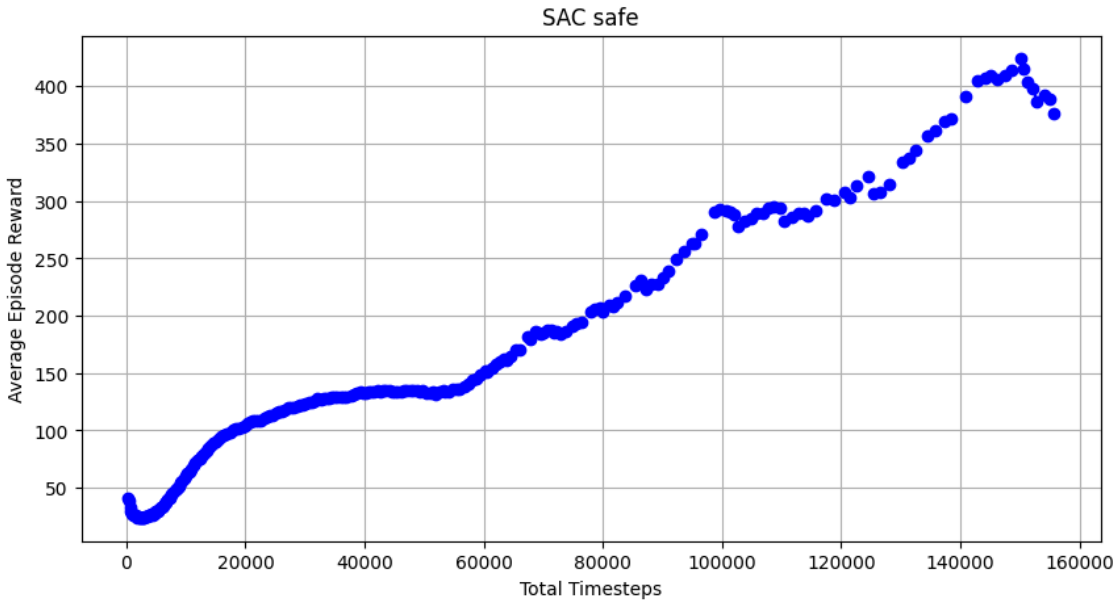
8. References

1. Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
2. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *ArXiv, abs/1801.01290*.
3. Holubar, Mario & Wiering, Marco. (2020). Continuous-action Reinforcement Learning for Playing Racing Games: Comparing SPG to PPO. https://arxiv.org/abs/2001.05270
4. Kuznetsov, Arsenii, et al. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics." International Conference on Machine Learning. PMLR, 2020.
5. E. Chisari, A. Liniger, A. Rupenyan, L. Van Gool and J. Lygeros, "Learning from Simulation, Racing in Reality," 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, pp. 8046-8052, doi: 10.1109/ICRA48506.2021.9562079. keywords: {Training;Smoothing methods;Automation;Conferences;Reinforcement learning;Predictive models;Trajectory}

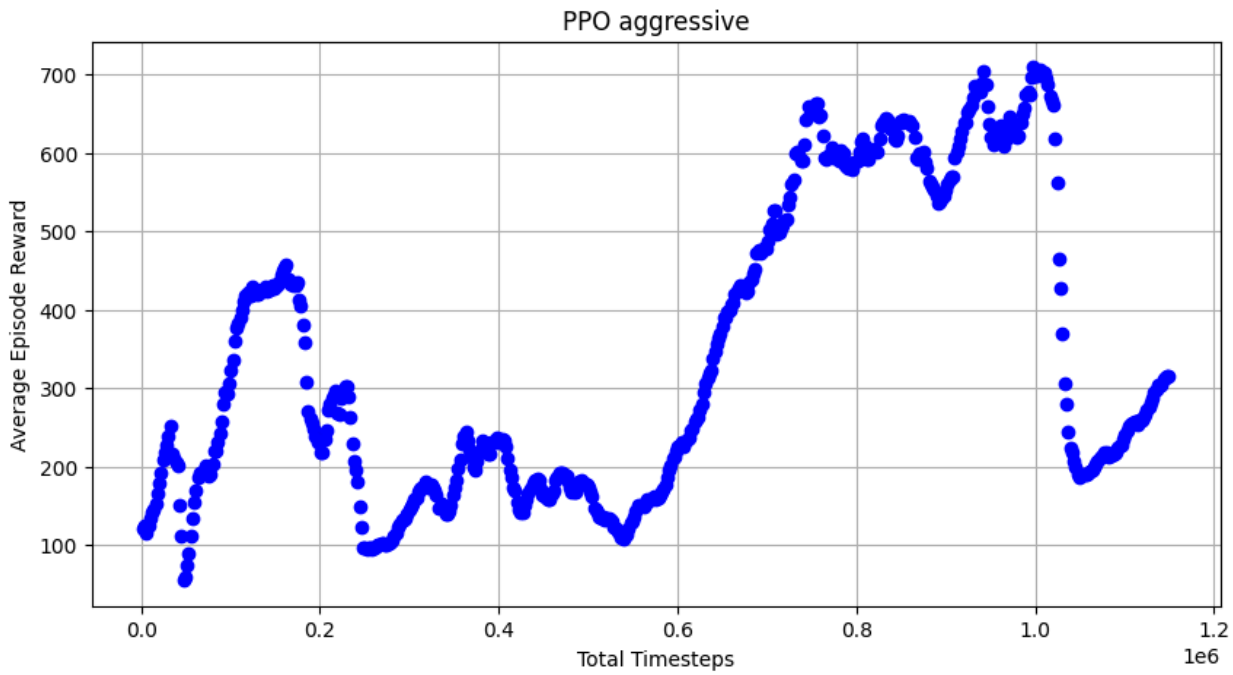
Appendix



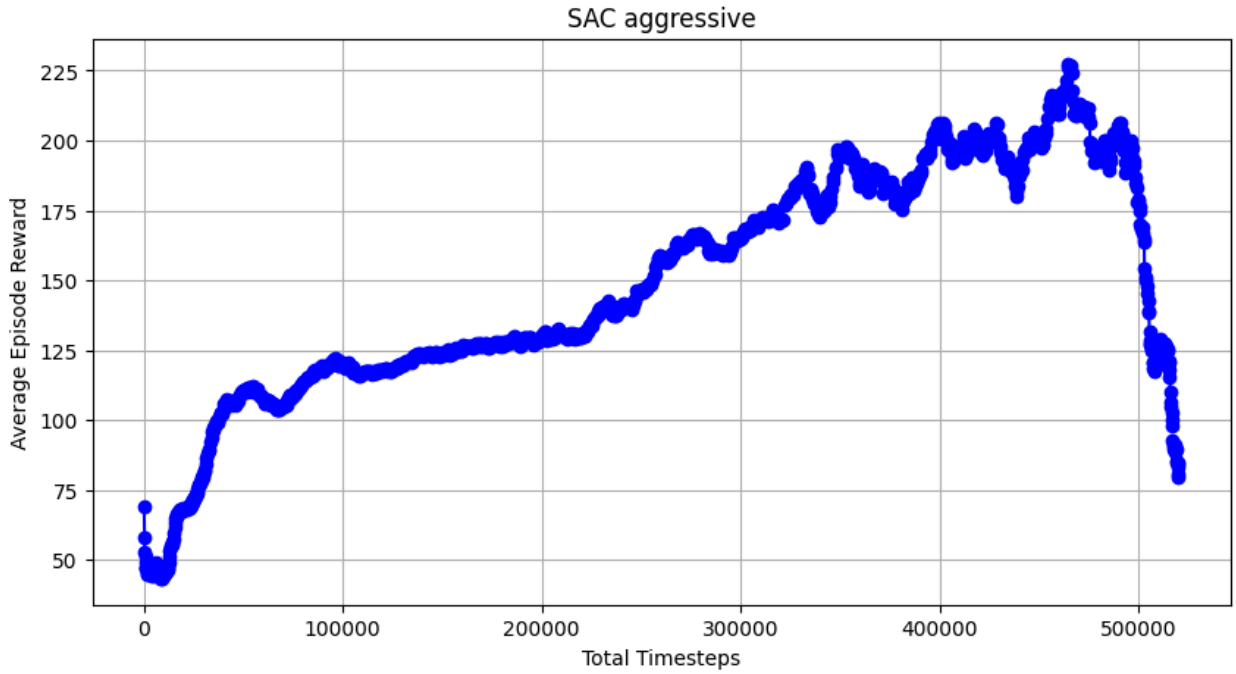
Training logs from the experiment to learn safe driving using PPO



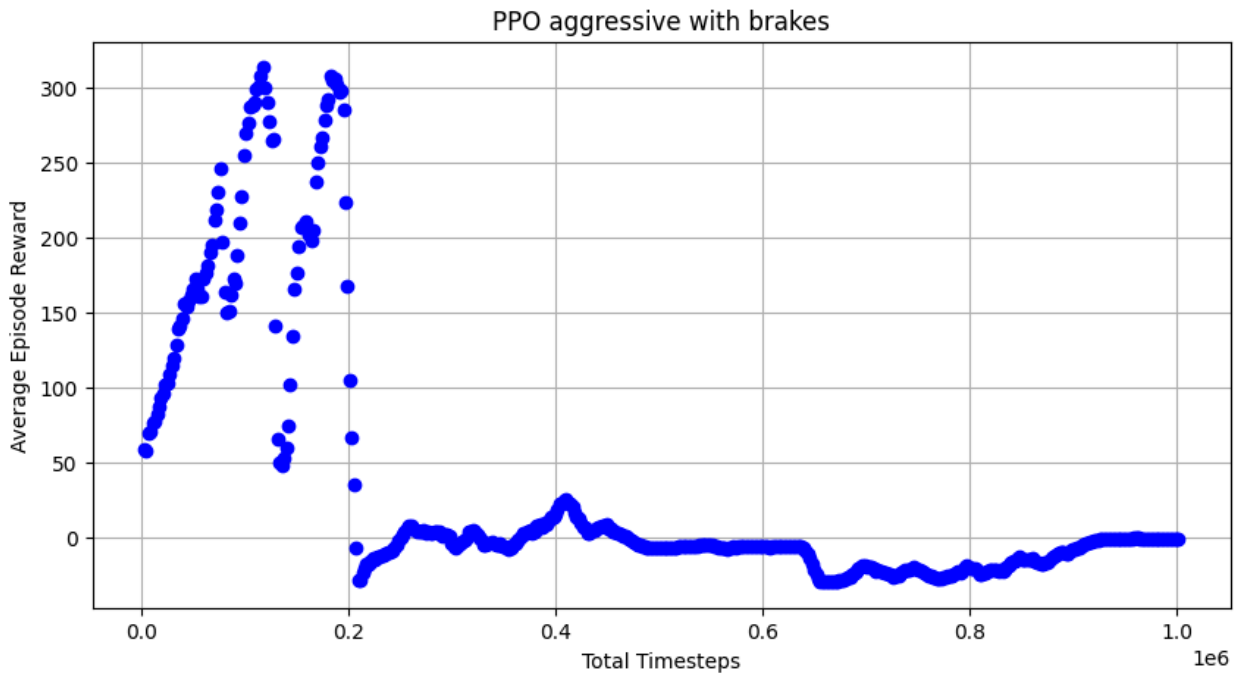
Training logs from the experiment to learn safe driving using SAC



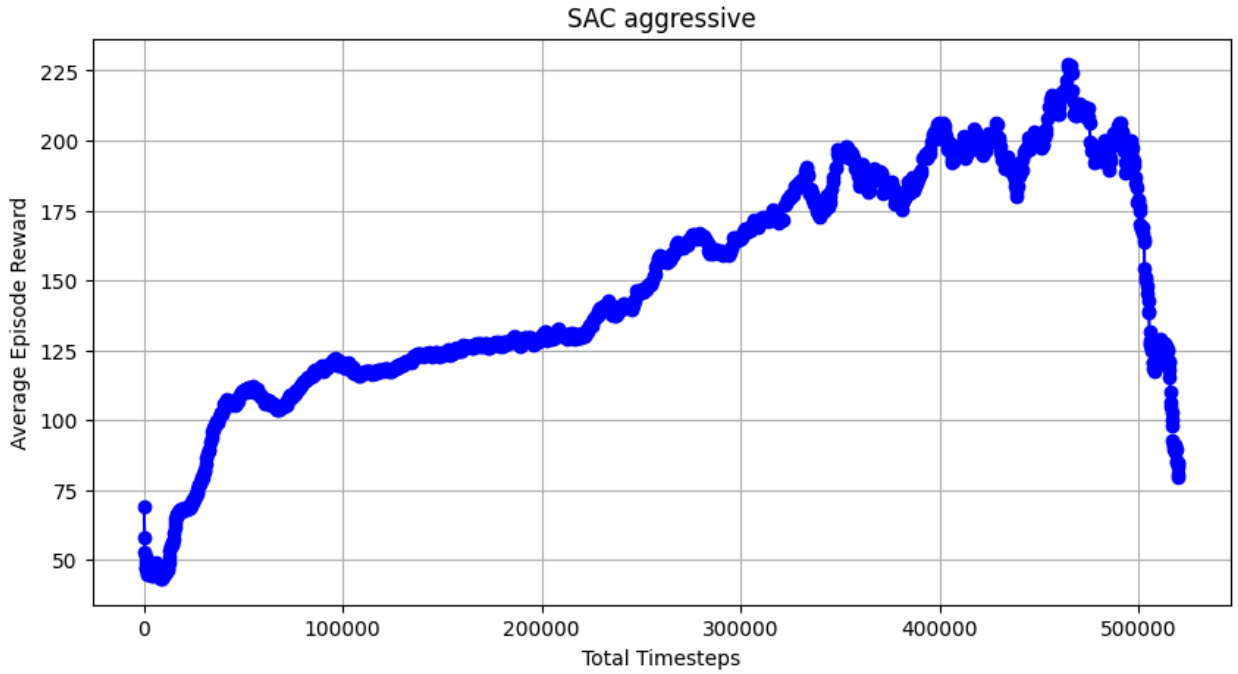
*Training logs from the experiment to learn aggressive driving using PPO**



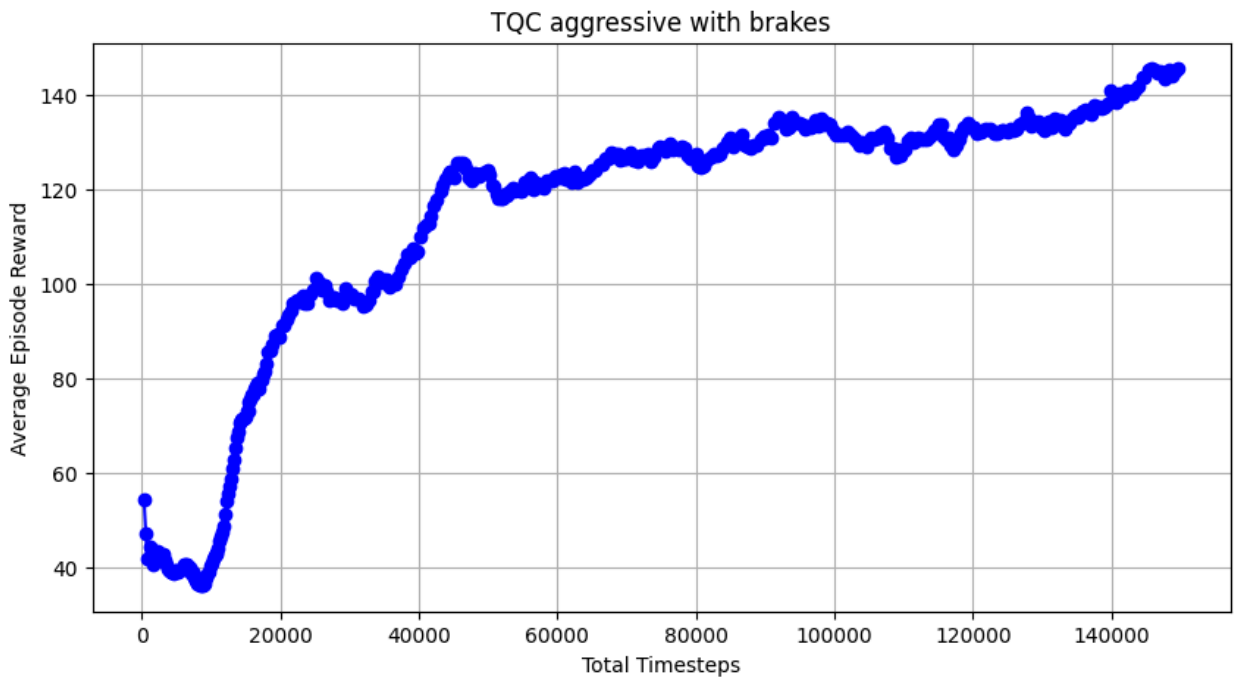
*Training logs from the experiment to learn aggressive driving using SAC**



*Training logs from the experiment to learn aggressive driving with braking using PPO**



*Training logs from the experiment to learn aggressive driving with braking using SAC**



Training logs from the experiment to learn aggressive driving with braking using TQC

***Note**: The sudden drop in performance occurs as the agent learns different ways to abuse the environment by driving over the finish line without completing the lap. This was either by learning to drive in circles or by driving in reverse.