ACAI: an Autonomous Racer
CSCI 8551

Archit Kalla
kalla100@umn.edu [1]

[1]University of Minnesota

December 2023

Abstract

This project aims to implement Reinforcement Learning techniques to develop an autonomous agent to navigate a racetrack. The environment used is Assetto Corsa, a racing simulation game that offers realistic vehicle handling in a racing environment. The goal is to train an agent capable of navigating a track leverages state of the art algorithms implemented in `stable-baselines3`. While the results were sub-optimal, the work done in this project lays the ground work for future experiments with more sophisticated methods and refined parameters. The code for this project can be found here https://github.com/archit-kalla/ACAI.

1 Introduction

With a race car there are many different variables that can affect a cars performance on track. Variables like tire temperature and pressure, aerodynamic surface setup, and suspension components can influence vehicle handling and thus, over the course of a race distance, can make the difference in overall race time. Race teams often employ sim drivers whose job is to drive and provide feedback on a race teams car in a simulator while the team analyzes the cars behavior on track ensuring all components stay within limits. Developing an AI driver capable to go quick around a circuit offers race teams the ability to highly parallelize the workflow and analyze far more data than one traditionally could where only one setup can be driven at a time by a sim driver. In addition, this can reduce workload on sim drivers as teams can give them more refined setups to provide feedback on. However, it is important to note that there are inputs that an AI agent can do that is outside of human capability and it is important for the race team to recognise this by analyzing the data and thus still requiring a sim driver to give the final say, because at the end of the day there is still a real human driving the race car on track.

1.1 Goals

The goal of this project is to train an agent to navigate Daytona International Speedway in a Lotus Exos 125 quickly. A reasonable time to complete the circuit is around 30 seconds so the goal with limited training resources and methodology is about one minute. The agent is prohibited from cutting the course by going through the grass and hitting a wall. In true oval racing fashion, any paved surface is fair game for the agent to drive on.

2 Related Work

Autonomous driving is a highly studied topic in the realm of autonomous agents, however the concept of a race speed capable agent is fairly now. We can turn to the realm of video games to find more examples of agents designed around racing. In the environment used in this experiment, the in built AI is far less sophisticated than the one proposed as it will take only the geometric line around a corner that has been hard-coded in the game file for each specific track. This technique still constitutes as an agent however a very limited one in terms of perception of the environment as the agent only knows what position on the track it is and how far from the line it is. Moving to more sophisticated implementations, iRacing, among the most popular online racing services, offers offline AI racing and while the internal working of the AI is proprietary the result is multiple agents that react dynamically to the environment and other vehicles to avoid contact and drive fast enough for close racing.

Moving away from games, in 2021 there was a championship called Robo-Race where electric cars race around real racetracks controlled entirely via AI software developed by the teams [7]. This shows that there is at least some interest in translating some of these techniques into the real world. Furthermore, F1TENTH is a project with the goal to take an RC car and modify it with robotic perception tools allow for the development of autonomous driving solutions [1].

Sony in their latest racing title Gran Turismo 7 have introduced their own AI racing by the name of GT Sophy. The goal stated by GT Sophy's creators state that the want the agent to be faster than a human. Their method is far more complex where they use a combination of policy networks optimization via reward functions and Soft Actor Critic (SAC) networks to control the agent [2]. Their approach also includes far more parameters in the observations made as this is a first-party project and thus have access to more information from the game itself without having to use more complicated methods such as computer vision to extract track edge data. An important note in the related work section is the authors comment on model free methods that while successful may not offer the fastest possible lap-time. Diving into their methodology, specifically, their reward and observation parameters. Part of the reward function in GT Sophy involves incorporating center-line progress and edge distance calculations where wall contact and edge breaches caused the reward to be reduced. However,

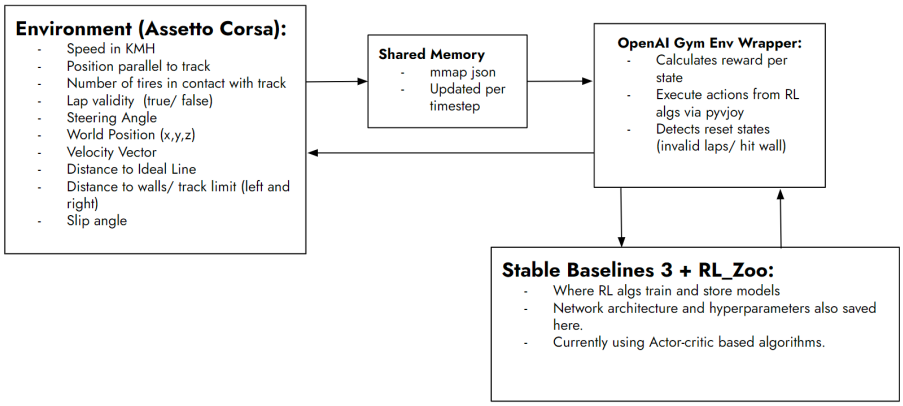

Figure 1: Methodology Overview

they note that they must add an energy component to the reduction of reward as the agent without this would just continuously grind against barriers as the center-line progress reward was overcoming the static reduction in reward. Another important note in their methodology is the following:

> The combination of throttle and brake in a single signal is motivated by an analysis of human recordings, which revealed that fast strategies do not involve any simultaneous use of throttle and brake.

This single signal brake-throttle will be used in this project and, in-fact, be taken even further to removing the brake altogether as there is no fast strategy around Daytona that requires brake input especially in the Lotus 125.

3 Method

The method proposed in the project involves wrapping our Assetto Corsa game environment with OpenAI's gym environment and then using state of the art model free reinforcement algorithms such as TD3 and SAC.

3.1 The Environment

Assetto Corsa offers a python app interface to extract some information from the simulation, so as part of this project a python app hooked into the simulator extracts information as see in Figure 1. This app creates a memory mapped JSON file to allow for high speed updates rather than waiting for disk write. Some information, however, is not readily available and had to be extracted manually. The ideal line, and distance to track limits on both left and right was extracted by running a lap exactly on each track limit and the ideal line, mapping the world coordinates to the normalized track position that is updated

Figure 2: Visual representation of the environment, purple represents geometric ideal line, red represents track limits, blue can be ignored

approximately every .5% of the track distance. It is important to note that the distance to track limits and ideal line is a perpendicular distance between the car's current position and the wall and does not have distance in front of the car as an observation. This was because the information was not readily available and computer vision methods that could approximate a distance would not have been reliable and added processing overhead that my machine could not handle.

Moving on to the track and car selection. For these experiments the track used is Daytona International Speedway, a 3.5 mile oval super speedway that to navigate quickly simply requires to pin the throttle and steer left on the turns. As for the car the Lotus Exos 125 was chosen as it is a lower spec Formula car with less power than Formula 1 yet quite a bit of down-force. The reduced power means that spikes in throttle input will not spin the rear tires and spin the car and the down-force makes the car responsive to turning even at high speed, thus making the Exos a very capable test car.

3.1.1 Gym Wrapper

The environment gym wrapper is responsible for calculating reward for each observations state and also responsible for executing actions back into the game. The reward function can be described simply as rewarding for track completion based off the distance completed between current state and the previous state. The reward is then discounted when the cars position is within 10 units of each track limit and scales based on distance (i.e. smaller distance means higher discount). Finally, the agent is rewarded based on its distance to geometric ideal line and scales with distance when within 7 units. The action execution

was done by using a virtual controller via `pyvjoy` where the inputs are steering angle between -1 to 1 and throttle between 0 to 1. Remember that there was no fast strategies that involve the brake around Daytona so the brake is omitted here. Reset states occur when the vehicle cuts a corner by going onto the grass or hits a wall. The car otherwise can touch any paved surface on the track in true oval racing fashion.

3.2 Reinforcement Learning Algorithms

The reinforcement learning algorithms used in this project are implemented via `stable-baselines3` [6]. Training loops are implemented via `rl_zoo3` [5]. Exact usage details are available on the README file here. In summary, `rl_zoo3` offers an easy to use training framework to execute training loops for algorithms implemented by `stable-baselines3`. Two experiments were conducted one using TD3 [3] and the other using SAC [4] algorithms. It is important to note that only one Assetto Corsa instance could be run at a time, thus sample efficiency during training would be important. These algorithms were used as they are highly sample efficient and they are compatible with the continuous action space that we are using. Other algorithms such as PPO were not used even though they are compatible with continuous action spaces they are highly sample inefficient. The hyperparameters were also set via `rl_zoo3` configuration files also described on the README file on the GitHub repository.

4 Results and Analysis

The results of the experiments were less than ideal. First as seen here the TD3 experiment was able to drive relatively straight on the main straight however on the first turn it takes a sharp left turn and drives right off the track. This was at around 70 thousand iterations however at more iterations there was no difference in progress. Around 100 thousand iterations the training takes around 5 hours to complete. Similarly SAC seen here happens to fall to the same fate instead turning right just before the start finish line. Obviously the project did not achieve the goal of navigating the track quickly much less actually completing a lap of the track. We can see obviously that the agent is over-fitted to going in a straight line. In this section we can observe some of the reasons and potential fixes for such issues, depending on feasibility.

4.1 Environment Analysis

To begin the spawn point for each iteration is the same, this reduces the variability and "track knowledge" that the agent can develop. As stated earlier it takes around 100 thousand iterations just to get down this first straight. The agent because of this is highly biased towards going in a straight line and to break out of this bias needed to be trained an introduced on multiple spawn points to get a better idea of the track. However, due to restrictions in what

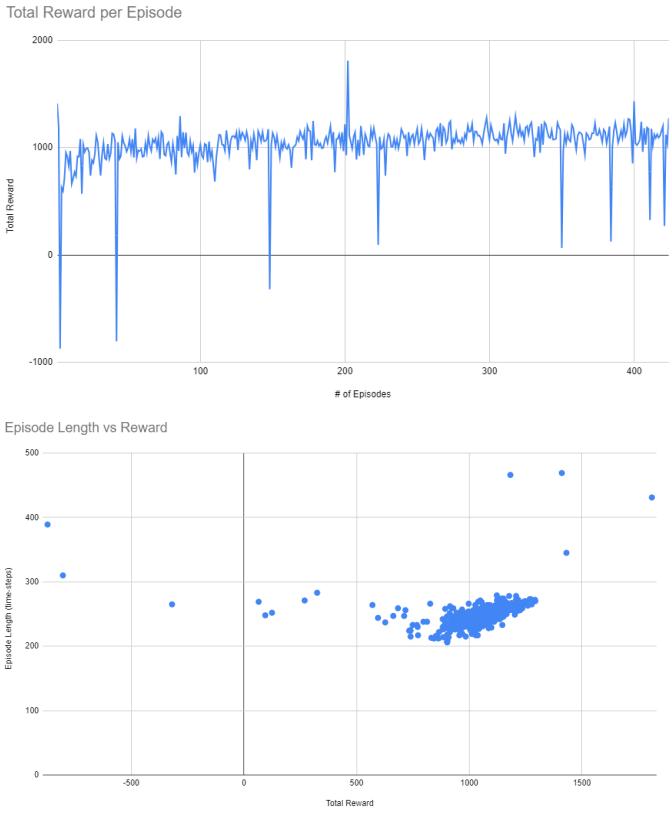

Figure 3: SAC: Total Reward per Episode(top) and Episode Length vs Reward(bottom)

can actually be changed in Assetto Corsa, without modifying game files heavily to allow for multiple spawn points this would be near impossible. Another solution to the constant spawn point problem could be to have a script run a preset set of actions to reach a random point on the track and then continue training from there. While this could be possible, to get this working consistently would require more time to fully flesh out.

Next, an obvious difference is the agents perception of its environment. As stated earlier, the distance to track limits measurements are perpendicular relative to the car (directly left and right) so the agent cannot "see ahead" this obviously limits this agents ability to navigate the course and take corrective action. This is likely the leading cause of the issues regarding the inability to turn as going in a straight line with limited perception is doable however turning requires more knowledge than just track limits at the sides.

Furthermore, the lack of parallelization is a limiting factor in training efficiency. On the current test machine, Assetto Corsa can only run one instance

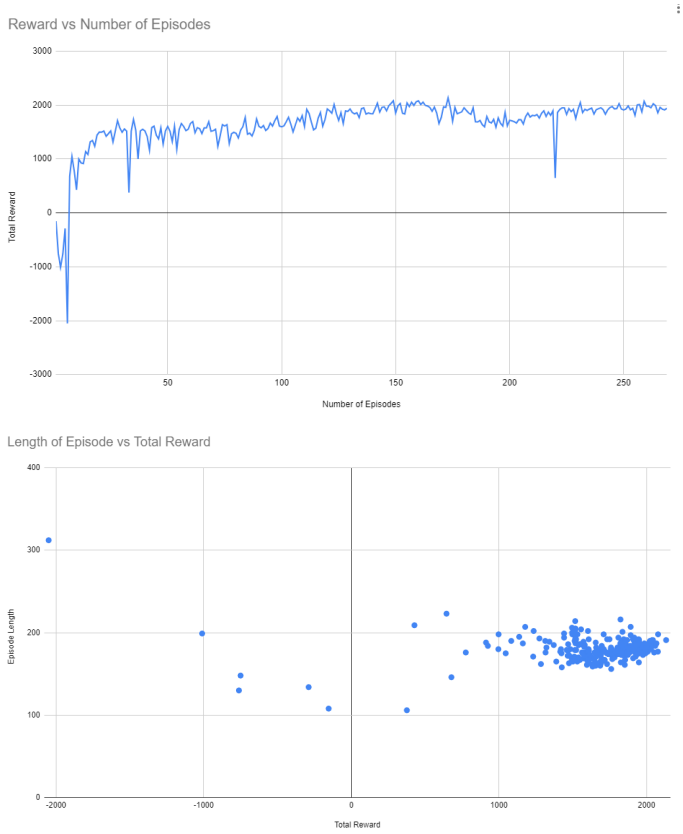

Figure 4: TD3: Total Reward per Episode (top) and Episode Length vs Reward (bottom) Reward scaling differs for TD3

with some level of stability. Comparing to the GT Sophy testing program where they have had 4 concurrent instances to train upon, the one instance of Assetto Corsa reduces the ability to get enough samples for training purposes. As seen in TD3's and SAC's papers, both algorithms support multiple instance training as part of the `stable-baselines3` implementation and as seen in their respective papers [3] [4] they can benefit greatly from the increase in sample size even though they are more sample efficient compared to other algorithms.

4.2 Algorithmic Analysis

Looking deeper into the algorithms chosen for this project SAC and TD3 both are variations of an actor critic based algorithm architecture. The actor network responsible for choosing the actions and the critic network responsible for evaluating the chosen actions from the actor network against the rewards from the state. There are some key differences between how SAC and TD3 work however

it is important to note that they both aim to solve the overestimation bias in different ways. First, to explain the over estimation bias in layman terms is when the algorithm has found a "good enough" solution to the problem space without fully exploring the space. In both algorithms they tackle this problem by introducing some sort of exploration to its action and also differences in how their policy networks update so that they avoid such overestimation problems. SAC introduces an entropy term to explore more of the environment by pseudo-randomizing the next output from the actor network. Changes increasing this entropy term encourages the actor network to select different actions to explore more of the environment. TD3 to achieve exploration introduces action noise at training time overlaid on the actor networks selected action, the strength of which scales the exploration of the algorithm. In terms of policy updates, SAC updates the policies by involving gradient ascent on the combination of the critic networks output and the entropy term and is done at every time-step [4]. Meanwhile TD3 does perform gradient ascent on the actor network based off the critic network, it does not update at every time step and rather delays the updates so more samples can be collected to help combat some of the over estimation bias that off policy algorithms can fall into [3].

These methods to combat overestimation bias both can achieve their goals with proper hyperparameter tuning. Learning rates are important hyperparameters that control the size of the step taken during the optimization process in both SAC and TD3. If the learning rate is too high, it can lead to instability and divergence, while a too-low learning rate can slow down learning or cause the algorithm to get stuck in local minima. Fine-tuning the learning rate is crucial for the stability and convergence of both SAC and TD3. Other parameters in both TD3 and SAC like Gamma and Tau which control discounting and target network updates also play important roles. Due to time constraints testing a variety of different hyperparameters was not possible hence why the results seen in Figure 3 and Figure 4 show that the agent overall was stuck in an local minima. However in 3 in both charts we see that as the number of episodes increase there is an very slow but increasing total reward per episode, so slowly the agent is learning more of the

5 Conclusions

While the results of the work so far have not shown much in terms of success towards the end goal, this project can serve as solid foundation to build upon to achieve the end goal. The architecture of the project enables for more input parameters to be added with little effort and when a new feature such as front of car perception is added the code can be modified with ease to enable a new agent training. Above in the analysis section we saw some of the pitfalls that this project had fallen into due to lengthy training time and lack of resources so having this platform to build off of without the added time pressure of a deadline can allow for the full goal to be realized. From previous work, it has been done to create an autonomous driving agent using model free methods, so this project has a strong future ahead with continued development.

5.1 Future Work

First, a major limiting factor to this project was the hardware that it was created on, a laptop. For Assetto Corsa to be able to run more than one instance, a more powerful system such as a desktop grade processor with a graphics card would go a long way in increasing stability. The multiple instances will allow for more samples and thus more efficient training. With the faster training time this should hopefully allow for a better understanding on how the hyperparameters are affecting the training of the agent and then this information can be used to tune the algorithms to suit this case the best. Next, adding functionality to consistently move the car around the track to random points is an important step to diversify training so the agent can get better at handling new sections of the course. Once the work to finish the vehicle navigating the course is complete the code to implement optimizing lap-time can be implemented. Other improvements include perhaps diversifying the training tracks from one track to multiple so that the agent can scale well with all tracks rather than just one track thus making the agent more robust and scale better.

References

[1] F1tenth website.

[2] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Dürr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, 2021.

[3] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[5] A. Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[6] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[7] RoboRace. Robrace website.