World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:19, No:3, 2025

Reinforcement Learning for Self Driving Racing Car Games

Adam Beaunoyer, Cory Beaunoyer, Mohammed Elmorsy, Hanan Saleh

Abstract—This research aims to create a reinforcement learning agent capable of racing in challenging simulated environments with a low collision count. We present a reinforcement learning agent that can navigate challenging tracks using both a Deep Q-Network (DQN) and a Soft Actor-Critic (SAC) method. A challenging track includes curves, jumps, and varying road widths throughout. Using open-source code on Github, the environment used in this research is based on the 1995 racing game WipeOut. The proposed reinforcement learning agent can navigate challenging tracks rapidly while maintaining low racing completion time and collision count. The results show that the SAC model outperforms the DQN model by a large margin. We also propose an alternative multiple-car model that can navigate the track without colliding with other vehicles on the track. The SAC model is the basis for the multiple-car model where it can complete the laps quicker than the single-car model but has a higher collision rate with the track wall.

Keywords—Reinforcement learning, soft actor-critic, Deep Q-Network, Self-driving cars, artificial intelligence, Gaming.

I. Introduction

REINFORCEMENT Learning (RL) is a branch of unsupervised machine learning where the learning agent takes actions in an environment and receives a reward based on the quality of the action [1]. Tools such as AlphaGo [2] and Sophy utilize this technology [3]. Alpha Go is an RL agent created by Google's Deep Mind, where the agent learns how to play the game Go autonomously and is able to beat professional players at the end of its training. With the current state of self-driving cars, there are numerous concerns about handling highly congested areas [4]. RL can be a solution for training self-driving cars, aiming to outperform humans, similar to how AlphaGo can now beat the best Go players in the world.

Using reinforcement learning agents to play video games is a growing area of research. Researchers developed agents to play the Flappy Bird video game using a Deep Q-Network (DQN) model. They found that different models yield varying performance results [5]. There is also a focus on training RL agents to control vehicles in simulated environments. A popular environment used for these agents is AWS DeepRacer since it provides users with a platform to design and develop an RL agent [6]. This service is commonly employed in studies focusing on object avoidance using different path-finding algorithms, such as A* and Line of Sight [7].

This research aims to train a reinforcement learning agent to race around a challenging track in car racing video games. The agent must complete the lap while maintaining quick lap times with minimal collisions. Prior studies investigate training an RL agent in a simulated environment with the goal of promoting high-speed racing without collisions [8]. In other studies, researchers use the Gran Turismo Sport game to train an RL agent to match the fastest human racers in a time trial [9]. In both previous studies, the RL agent is racing alone on realistic test racing tracks. To expand on previous research, this study will not only race with a single car but also consider scenarios with multiple cars racing on the same track. The track itself will also be more challenging than tracks in prior studies.

The following section provides a brief overview of Reinforcement Learning (RL). Section III explains the environment in which the RL agent is trained and introduces both the single-car race problem and the multi-car race problem. Section IV discusses the framework implemented for the single-car RL agent. Following this, Section V delves into the results of the single-car RL agent. Regarding the multi-car race problem, Section VI explains the RL agent multi-car framework and the changes made to expand from the single-car framework. Section VII breaks down the results of the multi-car framework. Finally, the conclusion section will summarize the research as well as discuss the future of this research.

II. Reinforcement Learning Overview

A few components are required to create a reinforcement learning agent. The first component is an action space. The action space defines the actions that the agent can take within the learning environment. There are two broad categories of action spaces: discrete and continuous. However, there are other categories of action spaces available and action space has major effects on the success of an agent [10]. The discrete action space defines discrete actions, meaning each action must be explicitly defined. A continuous action space allows the agent to choose an action based on a defined range. It can choose a value on a continuous range for more precise control of the environment [11]. The next component of the reinforcement learning agent is the observation space. The observation space defines the observations made on the environment. These observations form the basis for judging the agent's actions. Finally, the next component is the reward function for the reinforcement learning agent. This function takes the observations made and determines the total reward for the agent. A good reward function is critical when designing a reinforcement learning agent [12]. The reward

A. Beaunoyer, C. Beaunoyer, M. Elmorsy and H. Saleh are with the Department of Computer Science, MacEwan University, Edmonton, Alberta, Canada (e-mail: beaunoyera@mymacewan.ca, beaunoyerc2@mymacewan.ca, elmorsym@macewan.ca, haymourh0@macewan.ca).

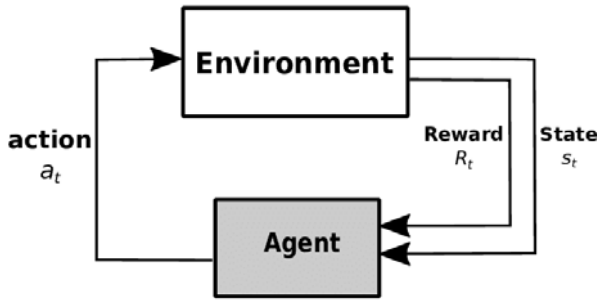


Fig. 1 Overview of Reinforcement learning

function undergoes several adjustments during model design and can significantly affect a model's performance.

A. RL Algorithms

Many RL algorithms, such as Deep Q-Network (DQN), Soft Actor-Critic (SAC), Advantage Actor-Critic (A2C), and others, can be used to train RL agents [13]. Each algorithm is unique, and the performance of each in the same environment can differ drastically. When implementing an RL agent, testing several algorithms is recommended, as some algorithms may perform very poorly in specific environments where others may excel. This paper will focus on two prevalent algorithms: DQN and SAC. DQN is a suitable algorithm since it excels in large state-action spaces and uses a deep neural network to determine the quality of the action taken by the learning agent. However, learning is limited to a discrete domain of actions [14]. RL agents commonly use the algorithm to play video games or even improve the game [15], [16]. SAC excels at learning rapidly in a limited sample size due to the algorithm encouraging more exploration. However, one of the critical features of SAC is that it allows the use of a continuous action space [17]. In prior studies, the SAC algorithm is used to control a bi-pedal robot in a simulated environment [18].

III. System Model

WipeOut (1995) is an arcade-style racing game that was the first in the series developed by Sony [19]. The base game provides users with various modes to play, such as time trials, single races, and grand prix. The game includes seven unique tracks, four racing teams, and eight pilots. Each team and pilot has different strengths and weaknesses. Some teams may have a higher top speed but need help to handle sharp turns. The game is known for its unique sci-fi setting, where each vehicle on the track is a futuristic ship that hovers above the track. The tracks include power-ups that can be collected and used by the player. There are also boost pads on the track that give players a significant speed boost. The players can steer the ship left and right, pitch the nose of the ship up and down, and thrust. The player also controls breaking left and right to help handle sharp corners. The player can also choose when to use their power-ups. The open source code replicating the WipeOut game can be found online on Github [20].

By employing socket programming, the WipeOut game is repurposed to serve as an RL training environment capable

Fig. 2 Wipeout Title Screen

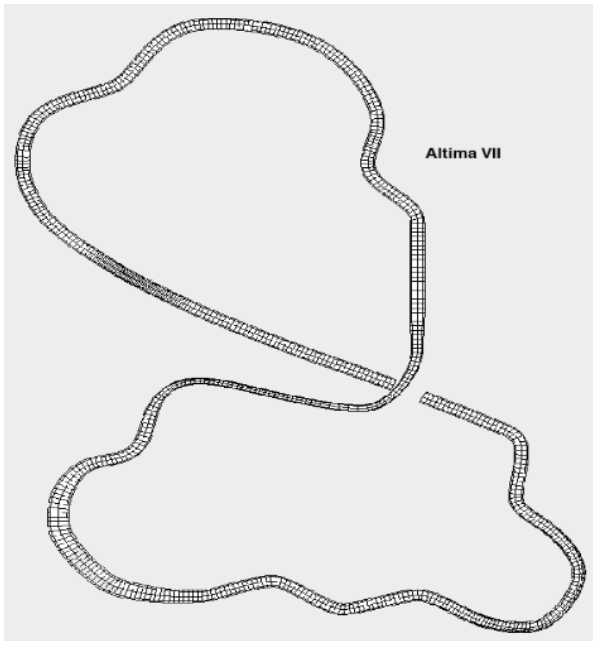


Fig. 3 Altima VII Track

of emulating an OpenAI Gymnasium environment [21]. The RL agent assumes control of the ship, which the player typically controls. Upon launching the application, the game is repurposed to run the time trial mode on the Altima VII track, automatically selecting pilot John Dekka.

A. Single Car Race Problem

The Altima VII track is intricate, with curves, varying track widths, and a jump (Fig. 3). The RL agent must navigate the track safely to keep the collisions with the track walls low and maintain low lap completion times. The varying track width also presents a unique challenge for the RL agent since it increases the chance of a collision when the track narrows. The jump is another unique obstacle the RL agent must adapt to. The RL agent will have no directional steering in the air. As a result, the RL agent will need to go on the jump at an angle that will ensure it stays on track when it lands.

B. Multi Car Race Problem

Sharing the road with another car adds another layer of complexity to the environment in which the RL agent must learn. The default AI controls the other car on the track. The default AI is on a designated path through the track and will make decisions throughout the race based on the distance to the player. The default AI may slow down if the player is too far behind or speed up if the player is too far ahead in an attempt to increase player engagement.

IV. Single Car RL Race Framework

In order to answer the single-car race problem of having competitive lap times while maintaining low collisions, there needs to be a framework in place. This framework requires implementing the Markov decision model, which requires a state space, an action space, and a reward function.

A. State Space

The state space of our RL implementation consisted of fourteen observations:

- Track section number - Reflects how far down the track the agent's car is.
- Lap time - Reflects how much time the agent is spending on a single lap.
- Distance to left wall - Determines how far the agent is from the left wall.
- Distance to right wall - Determines how far the agent is from the right wall.
- Car orientation - Determines if the agent is facing forward or backwards.
- Angle to the left wall - Determines the agent's angle to the left wall in the current section.
- Angle to the right wall - Determines the agent's angle to the right wall in the current section.
- Angle to the next left wall - Determines the agent's angle to the left wall in the next section.
- Angle to the next right wall - Determines the agent's angle to the right wall in the next section.
- Angle to the center of track - Determines the agent's angle to the center of the track (Fig. 4).
- Distance to the center of track - Determines the agent's distance to the center of the track.
- Thrust - Determines the agent's current thrust.
- Collision count - Determines the amount of collisions that occurred during a single lap.
- Distance to next section - Determines how far the agent is from the upcoming section.

B. Action Space

This research considers various RL methods, resulting in a varied action space implementation. The main actions available to the agent are similar to the basics of driving a standard car, such as thrust, brakes, and steering. Refer to RL methods for more details regarding the used action spaces.

C. Reward Function

The reward function implemented focuses on thrust, angle to the center, and distance to the center where thrust = T, angle to the center = θ_C, distance to the center = d_c, track width = t_w, and wrong direction degree = θ_W. For example, θ_W=180 when the car is parallel to the center of the track but facing the opposite direction.

Drawing inspiration from Kohsuke et al. [8], the following reward function is proposed for the single car problem:

$$R_t = \begin{cases} \mathrm{T} \times \left(\cos(\theta_C) - 2 \times |\frac{d_c}{t_w}|\right), \\ \theta_W \end{cases} \tag{1}$$

The reward function shown above resulted in the highest cumulative reward while maintaining a steady learning rate. This function aims to encourage speed and proper orientation to the center of the track. Since the RL agent is encouraged to stay close to the center of the track, the agent is less likely to collide with the wall. In summary, the reward function highly values speed as long as the risk of collision remains low.

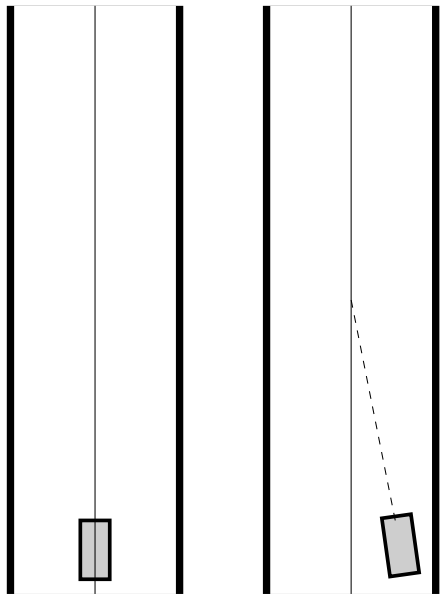
In the function, using the absolute value of the distance to the center is necessary because the center of the track equates to zero. Therefore, if the car is on the left side of the track, this would result in a negative number, and the right side would result in a positive number. The distance to the center is normalized by dividing it by the broadest section's width of the track; this is necessary since the tracks have varying widths. The distance to the center is weighted heavier than the angle to the center to prioritize being in the center rather than the angle to the center. The episode terminates and rewards the RL agent according to the terminating factors:

- Completed lap - If the agent completes the lap, the agent receives a large positive reward.
- Crashes - If the agent collides with the track 100 times, the agent receives a large negative reward.
- Wrong way - If the agent is facing the wrong way for 10 seconds, the agent receives a large negative reward.
- Time out - If the agent fails to complete the lap in under 30 minutes, the agent receives a large negative reward.

This research provides a framework for prioritizing the center of the track instead of the angle to the center. There is also the unique aspect of punishing if facing the wrong way based on the degree to which it is facing the wrong way. The most considerable punishment occurs when facing 180 degrees the wrong way. As the agent orientates itself in the correct direction, the punishment becomes less severe as the degrees go to zero and return to the correct direction. Previous research does not consider the degree to which a car is facing the wrong direction. Rather, the previous research only punishes the agent after a collision, or if the agent goes off track [8]. Furthermore, previous research does not prioritize the center of the track over the angle away from the track. Instead, the researchers prioritize the road ahead over being near the center of the track [8].

D. RL Methods

As mentioned earlier, action spaces may differ according to the chosen RL algorithm. In order to implement the DQN



(a) Ideal Path (b) An example of encouraged behavior for recovery

Fig. 4 Examples of angles to the center of the track

method, five discrete actions are available in the action space:

- Thrust - Accelerates the agent's car by the max thrust rate of the car.
- Turn right - Turns the agent's car right by the maximum turn rate.
- Turn left - Turns the agent's car left by the maximum turn rate.
- Brake - Decelerates the agent's car by the maximum brake rate.
- Coast - Allows the agent to take no action.

To implement the SAC method, two continuous actions are available in the action space:

- Thrust - Allows the agent to specify an acceleration rate between 300 to 600
- Turn rate - Allows the agent to specify a turn rate between -180 and 180 degrees

V. Single Car Race Results

In order to develop an RL agent that can minimize collisions while maintaining quick lap times, the environment needs a few adjustments. The ship's top speed is also reduced from 750 to 600 to ensure that collisions remain low during training but capable of fast lap times. With the StableBaselines3 python library [22], [23], the RL agent completes training on the Altima VII track.

Using DQN, the RL agent was unable to complete a single lap, either crashing too many times or going the wrong way

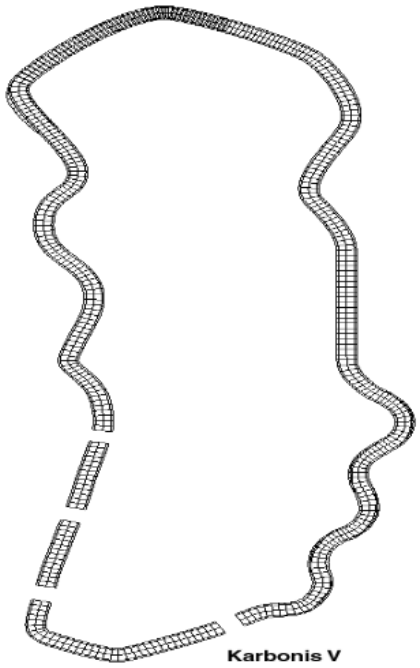


Fig. 5 Karonis V Track

(Fig. 6). Using the SAC model, the RL agent completes laps at a near 94% completion rate and crashes 6.52% of the time (Fig. 9). Furthermore, the total reward over time for the DQN model drops and shows no upward trend (Fig. 9). In contrast, the SAC model rises rapidly (Fig. 10). For the completed laps using the SAC model, the RL agent averages 179 seconds per lap. The fastest lap was 152 seconds. During the completed laps, the RL agent averaged 24 collisions with the track.

The SAC model trained for more episodes and completes the laps at a 95% completion rate (Fig. 7) while maintaining an upward trend in the total reward (Fig. 10). The SAC model produces an average lap time of 168 seconds, with the fastest lap being 146 seconds after extended training (Fig. 10). During these same laps, the average collisions by the RL agent is 13 collisions (Fig. 13).

With the fully trained SAC model, the RL agent raced on a new track Karbonis V (Fig. 5). On the new track, the trained agent completed laps at a 100% completion rate. During these laps, the agent averages a lap time of 114 seconds with the fastest lap being 106 seconds (Fig. 15). The average collisions experienced by the agent during these completed laps is 11 (Fig. 16).

A. Single Car Race Discussion

In the same amount of training, the SAC model outperforms the DQN model in quicker lap times and lower collision counts. The agent in the DQN model never completes a lap and will most likely crash too many times or go the wrong way (Fig. 6). If the agent does not complete a lap within 30 minutes it will trigger the timeout condition and if it faces the wrong way for ten seconds it will trigger the wrong way condition. However, when compared to SAC, the agent in the SAC model

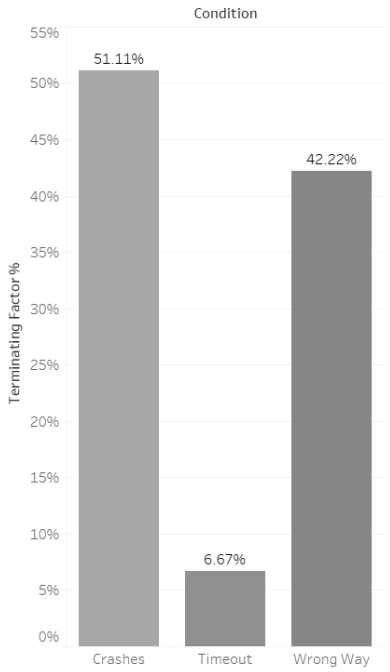


Fig. 6 Terminating condition of DQN training

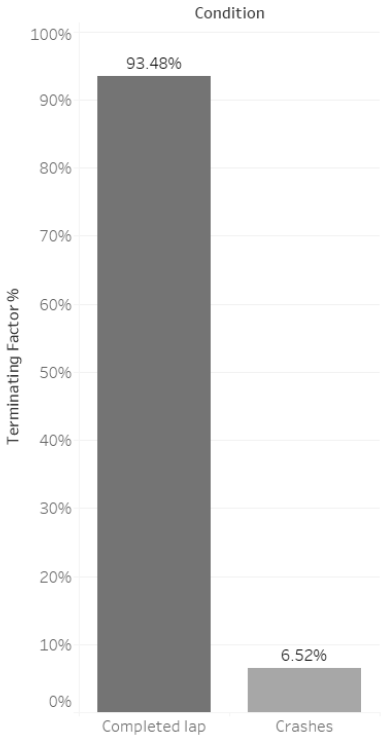


Fig. 7 Terminating condition of SAC training

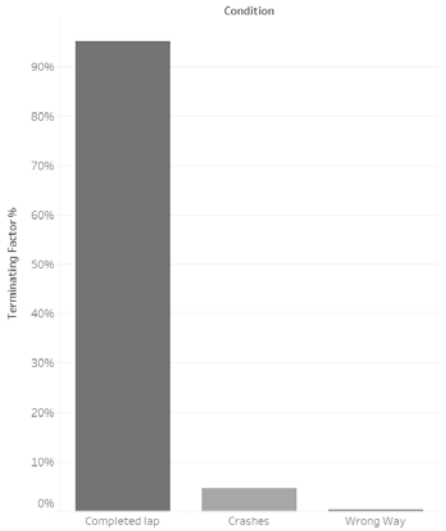


Fig. 8 Terminating condition for SAC model after longer training

never times out or goes the wrong way (Fig. 8). This is likely due to the SAC model not being able to thrust below half their max thrust. The agent in the SAC model was also able to crash less than the DQN agent (Fig. 7). Once again, the action space for the SAC model allows more precise control of the steering, allowing the agent to stay closer to the center than the DQN agent is capable of. In addition, the continuous action space allows the agent to see better success in this environment.

It is clear that the SAC model outperforms the DQN model in this environment. Therefore, the SAC model is the model that will undergo more training. The RL agent trained for six million time steps and was able to complete laps at a 95 percent completion rate (Fig. 10). Along with a high completion rate (Fig. 10), the agent maintained relatively fast lap times (Fig. 12). The track record is 86 seconds, which equates to the agent being 41% slower than the track record. However, it is worth noting that the agent's speed was reduced by 20%. For each completed lap, the agent can complete the lap with an average of thirteen collisions with the walls and ground. In 14% of the completed laps, the agent completed the lap without a single collision. It is worth noting that the jump in the Altima VII track will often cause the ship to bounce and hit the ground three times, which counts as a collision. If the ship lands at the perfect angle and speed, there will be zero collisions. These episodes are captured during the agent's training; therefore, the expected averages are expected to improve if only the data from the fully trained model are considered.

In order to test the robustness of the agent, the fully trained agent races on a new track, Karbonis V (Fig. 5). The agent completes 100 episodes on the new track with a 100% completion rate. The track record was 56 seconds, meaning the agent is on average, 58 seconds slower than the track record. The average collision count is eleven, and in 3% of the episodes, the agent completed the lap with no collisions. Since the new Karbonis V track includes four jumps, high collisions are expected. As previously stated, collisions will be associated

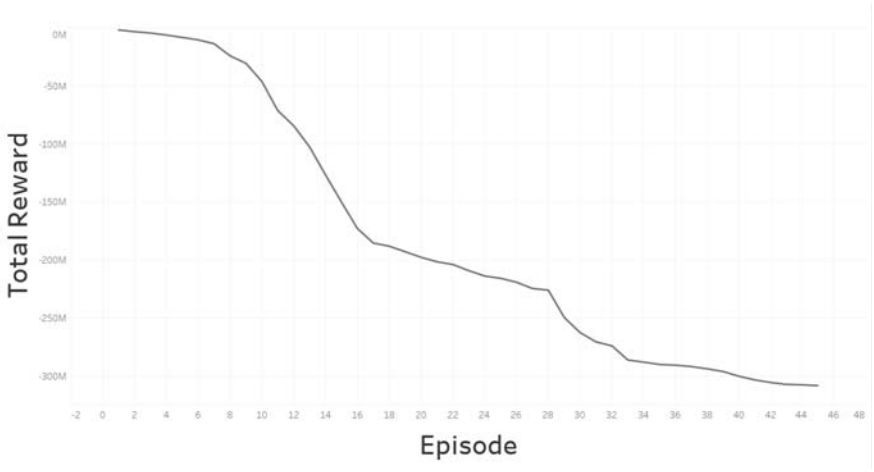


Fig. 9 Total reward over time of DQN training

Fig. 10 Total reward over time of SAC training

Fig. 11 Total reward over time of SAC after longer training

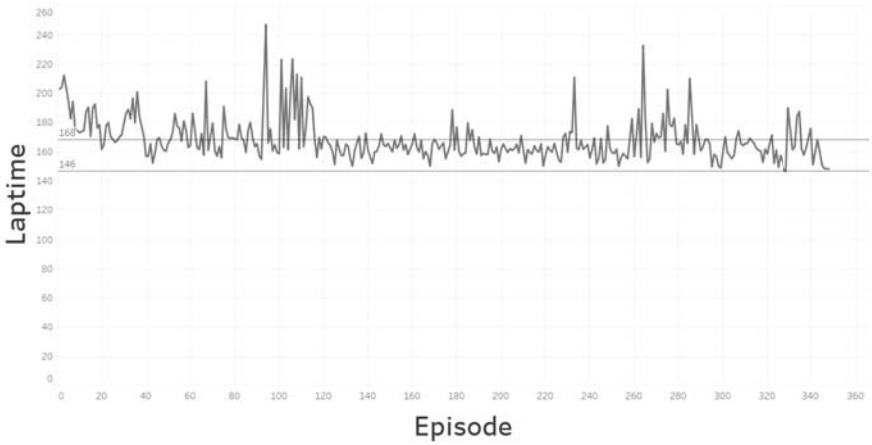


Fig. 12 Lap times of completed laps of SAC during extended training

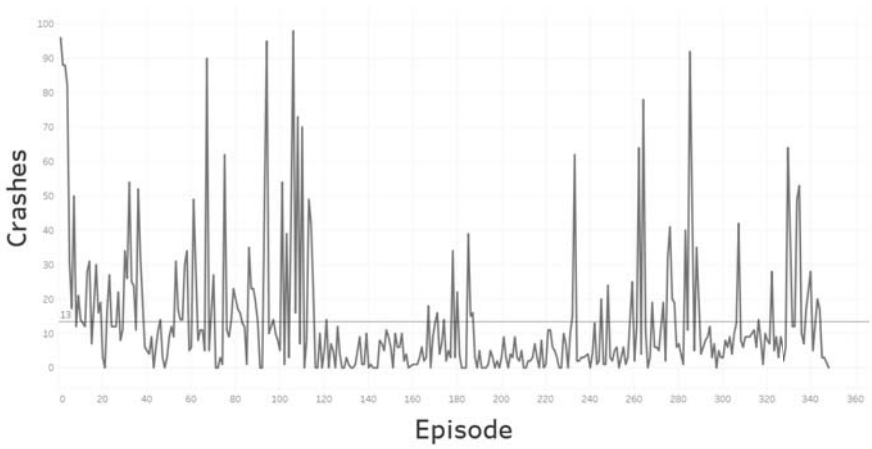


Fig. 13 Total crashes for a completed lap of SAC during extended training

Fig. 14 Total reward over time for trained SAC model on Karonis V

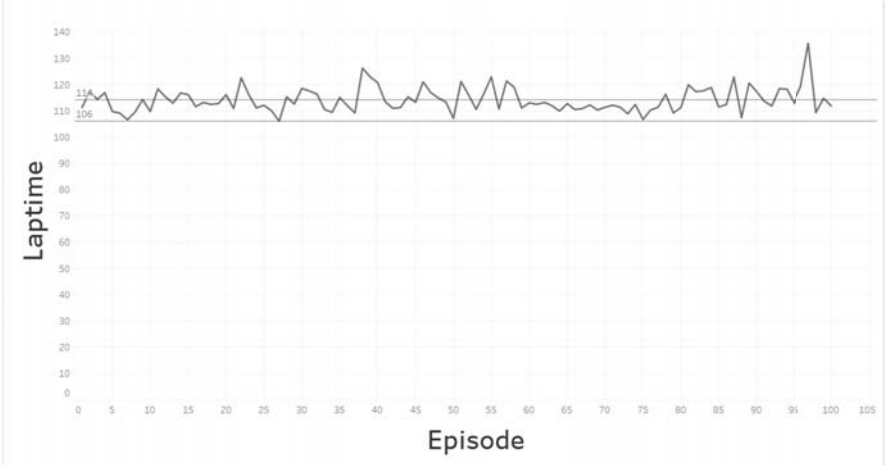


Fig. 15 Lap times of completed laps for trained SAC model on Karonis V

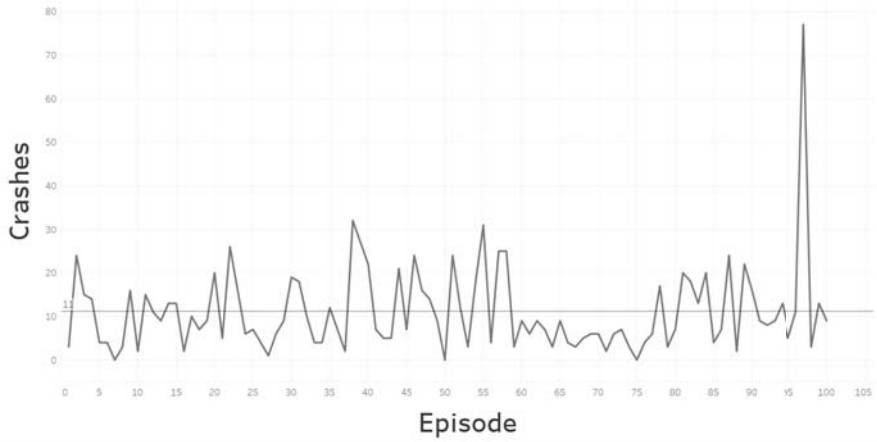


Fig. 16 Total crashes during a completed lap for trained SAC model on Karonis V

with each jump unless the jump is perfectly executed. The fully trained model allows the RL agent to maintain competitive lap times while keeping a low collision count on any track. This balance between speed and safety is crucial for the RL agent's performance.

The leading cause for the agent failing to beat the track record is the reduced speed and goal of staying near the center of the track. In order to be quick in a lap, the driver must use the boost pads on the track, which are often located on the sides of the track not the center. Furthermore, in racing there is often an optimal path for speed, which involves being close to the walls in the apex of a curve, since the RL agent prioritizes being closer to the center, it is off the racing line. The track record also requires no collisions throughout the track.

VI. Multi-Car Race Framework

The original single-car SAC framework is adapted to address the multi-car race problem, which entails sharing the road with another car while maintaining competitive lap times with minimal collisions. With the added complexity of sharing the track, several new observations emerge, leading to modifications of the reward function. This implemented framework continues to use the SAC model, and the action space remained the same.

A. State Space

All of the previous observations remain in place, however four new observations are made:

- Angle to other car - Determines the angle difference between the agent's car and the other car (Fig. 17).
- Section difference - Calculates the difference between the sections of the agent's car and the other car.
- Collisions with other car - Calculates the amount of the times the agent and other car collided.
- Distance between cars - Calculates the total distance between the agent's car and the other car.

B. Reward Function

The proposed reward function continues to focus on thrust, angle to the center, and distance to the center with some additions.

$$R_t = \begin{cases} \mathrm{T} \times \left[\left(\cos(\theta_C) - 2 \times |\frac{d_c}{t_w}| \right) + \left(\frac{d_b}{100} \times \cos(\theta_B) \right) \right], \\ \theta_W \ . \end{cases} \tag{2}$$

where thrust = T, angle to the center = θ C, distance to the center = d_c, track width = t_w, distance between cars = d_b, angle between cars = θ_B, and wrong direction degree = θ_W.

The goal of this function is to continue to encourage speed and driving in the center of the track while also considering the position of the other car. The distance between cars is used as a punishment and reward. Since the distance results in a negative number if the other car is ahead of the agent. However, if the agent is ahead, it will receive a positive reward based on the distance. If the agent is behind the other car, the distance is negative. Otherwise, it is positive. This distance is then multiplied by the angle between the cars. The purpose of multiplying it by the angle is to promote driving parallel with the other car instead of crossing paths, which could result in a collision. It is important to note that the distance between cars is divided by 100 to scale it better with the equation. The distance between cars proliferates, which may result in an imbalance in the reward. There are also more terminating factors to reflect the multi-car model. In addition to the single car terminating factors, the multi car model can also terminate if:

- Too far behind - If the agent falls 50 sections behind, the agent receives a large negative reward.
- Collide with other car - If the agent ever collides with the other car, the agent receives a large negative reward.

VII. Multi-Car Race Results

In order to develop an agent that can navigate safely and rapidly through a track while avoiding another car, the environment requires further modifications. The game is modified to track and calculate the angle between the two cars' forward vectors. The game now tracks collisions made with other cars specifically while keeping collisions made with the track separate. The game now tracks the distance between the two cars and the section each car is in. The top speed of the agent's ship is returned to the default top speed to ensure it can keep up with the default AI's ship. Once again, the agent trains on the Altima VII track (Fig. 3) using StableBaselines3 [21] as the library to provide the training tools.

With the two cars on the track, the agent completes 59% of laps while crashing 41% of the time with the wall (Fig. 18). The total reward also was on a downward trend; however, halfway through the training, the results began trending upwards (Fig. 17). The agent completes a lap with an average lap time of 161 seconds, with a fastest lap of 133 seconds (Fig. 20). On these laps, the average collision count is 53 with the lowest amount of collisions being 4 (Fig. 21).

A. Multi Car Race Discussion

During training, the agent completes the majority of laps; however, there are many episodes where the agent crashes too many times (Fig. 18). With the speed increase, there

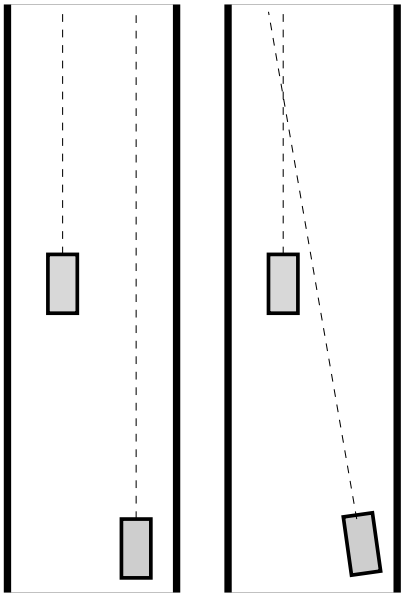


Fig. 17 Angle to other car (Driving parallel is encouraged, crossing paths is discouraged)

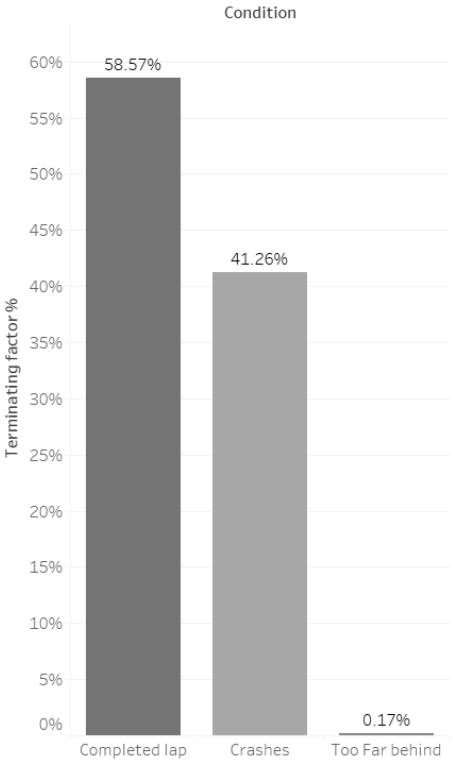


Fig. 18 Terminating condition for Multi car SAC model on Altima VII during training

Fig. 19 Total reward over time for Multi car SAC model on Altima VII during training

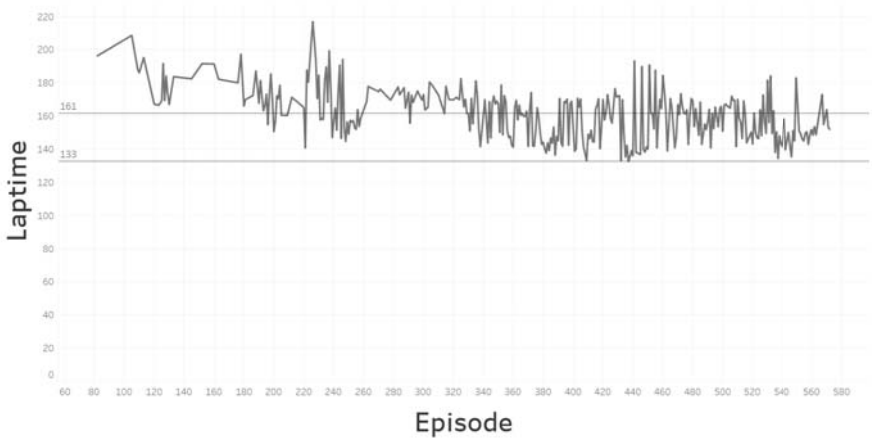


Fig. 20 Lap times of completed laps for Multi car SAC model on Altima VII during training

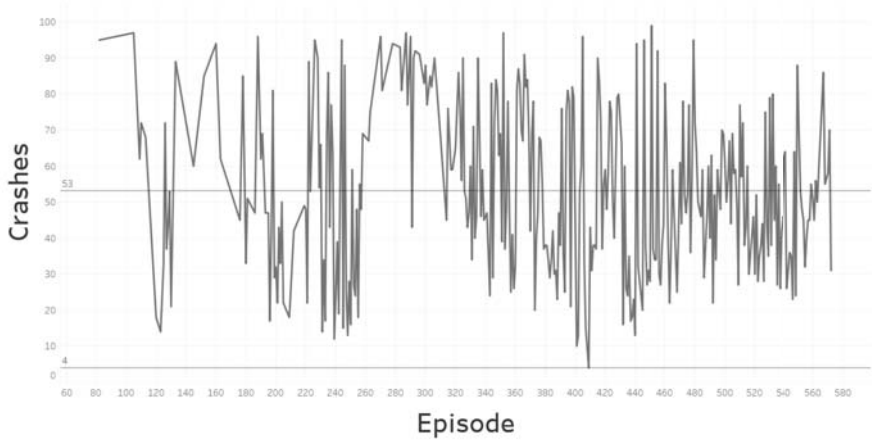


Fig. 21 Total crashes during a completed lap for Multi car SAC model on Altima VII during training

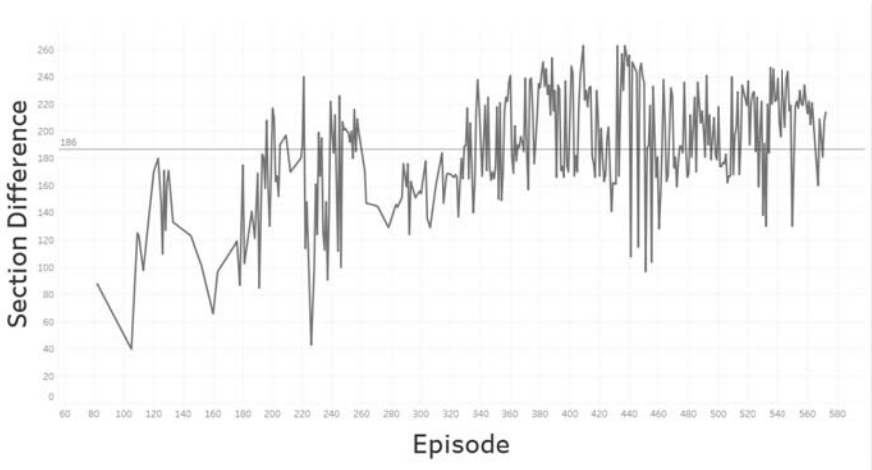


Fig. 22 Section difference at the end of a completed lap for Multi car SAC model on Altima VII during training

is a higher likelihood of the agent colliding with the track. The overall reward over time also reflects the early training sessions, where the agent crashes too many times more often than completing a lap (Fig. 19). However, the upward trend displays that after sufficient training, the agent becomes far more likely to complete the lap rather than crash too many times. Also, it is worth mentioning that the agent on average 186 sections ahead of the default AI during a completed lap, which is on a determined path (Fig. 22). When the RL agent crashes too many times, it could eventually be left behind; however, most times, the agent still completes the lap (Fig. 18). Most importantly, there is never a case where the two cars collide. If the cars collide once the episode terminates, however, there was never a case of that being the terminating condition (Fig. 18). This indicates that the RL agent did a good job avoiding the other car on the track.

When the agent completes a lap, it has a faster average lap time than the single-car SAC model and a better fastest lap. The multi-car agent was, on average, 4% faster, and when comparing their fastest times, the multi-car algorithm was 9% faster. These results reflect the faster top speed used by the multi-car model. However, due to the increased speed, the average collision on a completed lap for the multi-car model is 121% higher, which slows learning in the multi-car model.

The top speed is a critical factor in safe driving. However, the modified reward function encourages being ahead of the other car. The further ahead the agent is, the better the reward will be. Since the two vehicles have the same top speed, once the agent can navigate the track, the two cars are usually in the same section of the track. Since the cars are usually near each other, this eliminates the potential repercussions of calculating the angle between cars in different track positions.

VIII. Conclusion and Future Work

The research provides a foundation for training an RL agent to navigate challenging tracks with relatively limited observations. The RL agent can complete the lap by itself and perform well with other cars on the track. The single-car model is capable of completing laps consistently and with few collisions. As seen in Fig. 18, in the multi-car model, the agent never collides with the other car and can complete the lap faster than the default AI, which is on a determined ideal path. Therefore, the SAC model finds a faster path through the track and avoids other vehicles at all times.

Future work will consider continuous reward function improvement for the multi-car framework. Another future direction is using reinforcement learning for solving other games and self driving real vehicles.

References

[1] R. Sutton and A. Barto, Reinforcement Learning An Introduction Cambridge, MA: The MIT Press, 2018, [E-Book] Available: http://incompleteideas.net/book/RLbook2020.pdf

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, and M. Leach, "Mastering the game of go with deep neural networks and tree search", *Nature*, vol. 529, no. 7587, 2017, doi: 10.1038/nature16961

[3] P. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, and H. Kitano, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature*, vol. 602, pp. 223-228, 2022, doi: 10.1038/s41586-021-04357-7.

[4] N. Wang, X. Wang, P. Palacharla, and T. Ikeuchi, "Cooperative autonomous driving for traffic congestion avoidance through vehicle-to-vehicle communications," *IEEE Vehicular Networking Conference (VNC)*, pp. 327-330, 2017 doi: 10.1109/VNC.2017.8275620.

[5] Y. Sun, "Performance of Reinforcement Learning on Traditional Video Games," *3rd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, pp. 276-279, 2021 doi: 10.1109/AIAM54119.2021.00063.

[6] L. A. Garza-Coello, M. M. Zubler, A. N. Montiel and C. Vazquez-Hurtado, "AWS DeepRacer: A Way to Understand and Apply the Reinforcement Learning Methods," *IEEE Global Engineering Education Conference (EDUCON)*, pp. 1-3, 2023, doi: 10.1109/EDUCON54358.2023.10125166.

[7] J. McCalip, M. Pradhan and K. Yang, "Reinforcement Learning Approaches for Racing and Object Avoidance on AWS DeepRacer," *IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 958-961, 2023, doi: 10.1109/COMPSAC57700.2023.00129.

[8] T. Kohsuke, S. Yuta, O. Yuichi and S. Taro, "Design of Reward Functions for RL-based High-Speed Autonomous Driving," *IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)* pp. 32-37, 2022, doi: 10.1109/MCSoC57363.2022.00015.

[9] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Durr, "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning." *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021, doi: 10.1109/LRA.2021.3064284

[10] A. Kanervisto, C. Scheller and V. Hautamäki, "Action Space Shaping in Deep Reinforcement Learning." *IEEE Conference on Games (CoG)*, Osaka, Japan, pp. 479-486, 2020, doi: 10.1109/CoG47356.2020.9231687.

[11] Y. Arjoune and S. Faruque, "Double Deep Q-Learning and SAC Based Hybrid Beamforming for 5G and Beyond Millimeter-Wave Systems." *IEEE International Conference on Electro Information Technology (EIT)*, pp. 422-428, 2021, doi: 10.1109/EIT51626.2021.9491918.

[12] H. Goh, Y. Huang, C. Lim, D. Zhang, H. Liu, W. Dai, T. Kurniawan, and S. Rahman, "An Assessment of Multistage Reward Function Design for Deep Reinforcement Learning-Based Microgrid Energy Management." *IEEE Transactions on Smart Grid*, vol. 13, no. 6, pp 4300–4311, 2022, doi:10.1109/TSG.2022.3179567

[13] C. Lazaridis, I. Michailidis, G. Karatzinis, P. Michailidis, and E. Kosmatopoulos, "Evaluating Reinforcement Learning Algorithms in Residential Energy Saving and Comfort Management," *Energies*, vol. 1, no. 2, pp. 581-614, 2024, https://doi.org/10.3390/en17030581

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint* arXiv:1312.5602, 2013 doi: 10.48550/arXiv.1312.5602

[15] Y. Dong, "Research on deep learning video game simulation algorithm based on Western Wind Music," *Entertainment Computing*, vol. 46, p. 100566, May 2023. doi:10.1016/j.entcom.2023.100566

[16] N. A. Barriga, M. Stanescu, F. Besoain, and M. Buro, "Improving RTS game AI by supervised policy learning, tactical search, and deep reinforcement learning," *IEEE Computational Intelligence Magazine*, vol. 14, no. 3, pp. 8–18, Aug. 2019. doi:10.1109/mci.2019.2919363

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." *In International conference on machine learning*, pp. 1861-1870, 2018, doi: 10.48550/arXiv.1801.01290

[18] O. Aydogmus and M. Yilmaz, "Comparative analysis of reinforcement learning algorithms for Bipedal Robot locomotion," *IEEE Access*, vol. 12, pp. 7490–7499, 2024. doi:10.1109/access.2023.3344393

[19] "Wipeout." *Psygnosis, published by Sony Computer Entertainment*, 1995.

[20] D. Szablewski, "wipeout-rewrite." *Github repository*, https://github.com/phoboslab/wipeout-rewrite

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, and J. Schulman, "OpenAI Gym." *OpenAI*, 2016. [Online]. Available: https://github.com/openai/gym.

[22] A. Raffin, A. Hill, A. Gleeve, A.Kanervisto, and M. Ernestu, "Stable Baselines3 : Reliable Reinforcement Learning Implementations." *Journal of Machine Learning Research*, vol. 22, no. 268. 1-8, 2021. http://jmlr.org/papers/v22/20-1364.html

[23] S. Sivashangaran, A. Khairnar, and A. Eskandarian, "AutoVRL: A high fidelity autonomous ground vehicle simulator for SIM-to-real deep reinforcement learning," *IFAC-PapersOnLine*, vol. 56, no. 3, pp. 475–480, 2023. doi:10.1016/j.ifacol.2023.12.069