ME5406: Deep Learning for Robotics
Project 2: Optimised Autonomous Racing

Name: Dinesh S/O Magesvaran
Matriculation Number: A0182885M
Email: E0309680@u.nus.edu

Introduction

In racing, there is an optimal racing line that race car drivers strive to follow to complete the race in the shortest time possible. However, as racing tracks can be complex, the optimal racing line will differ from track to track. In races such as F1, the race car drivers spend a lot of time driving in F1 training simulators to improve their timings[1]. These simulators can allow the drivers to manually determine the best path that they can take along the track to clock the fastest timings. Although their vast experience in driving will enable them to easily determine near optimal racing lines along the track, it may not be the most optimal. Even if the driver takes a path that is only slightly less optimal than the best path, it may add seconds to the race time, which could result in the driver losing the race. Machine learning can be used to aid the drivers in learning the best paths through simulation, so that the race car drivers can maximise their performance.

Proposed Method

To solve the problem of finding the optimal racing path on the track, our team has proposed the use of Deep Reinforcement Learning.

State Space

The state space consists of a continuous space of the possible x, y and angle coordinates of the car, which is the agent. Each state will also correspond to a particular image that is rendered by the environment graphics. This image will be fed as input to the neural network.

Action Space

The action space consists of a discrete space of 5 possible actions that the agent can take, with each action giving a value for steering, acceleration and brake. Table 1 below shows the actions in detail.

Table 1: The 5 possible actions for the agent

Action	Steering	Acceleration	Brake
1	-1	0	0
2	1	0	0
3	0	0	0
4	0	1	0
5	0	0	0.5

Reward Structure

The reward structure consists of the rewards given to the agent after it has performed an action, depending on how desirable the outcome is towards finding an optimal racing path on the track. Table 2 below shows the details of the reward structure.

Table 2: Reward structure for the agent

Outcome	Reward
Crossed Finish Line	+10,000
Went Out of Track	-10,000
Stopped Moving	-10,000
Time Limit Exceeded (set at 500s)	-10,000
Time Penalty (given every time step of 0.1s)	-0.1
Crossed Checkpoint (3 checkpoints on track)	+1,000
New Distance travelled on track (Straight segment every metre)	+1
New Distance travelled on track (Curved segment every radian)	+100

Initially, the reward structure was simpler, and more components were added to improve performance. ‘Crossed finish line’ and ‘Went Out of Track’ were added from the start. As there was no incentive during training for the agent before it reached the finish line, 3 checkpoints were added with positive rewards. For further incentives, positive rewards were added based on new distance along the track travelled by the agent. As the agent managed to complete the track, it was taking its time and moving very slowly. Thus, the time penalty was added as a negative reward given at every time step. Exceeding the time limit also now has a negative reward given.

Neural Network

The neural network consists of a Convolution Neural Network which takes in the image data from the environment graphics and undergoes max pool operations to process the image. The output is flattened and fed into a dense layer, which is responsible for learning the weights and biases for estimating q-values for the 5 possible actions based on the given state images. The output is used by the agent to determine the best action that it can take based on what the neural network model has learned.

Code Implementation

For the code implementation, the race car environment was fully coded from scratch, with no code taken from online. The environment consists of RaceCarEnv.py, which is an OpenAI Gym custom environment, and Graphics.py, which is the graphics file for performing rendering operations for the environment. Graphics.py uses Pygame and OpenGL libraries/packages.

For the agent and neural network, the code was taken from an online GitHub repository that was written for a similar race car problem[2]. Most of the code was usable for our problem. Some modifications were made to the code to make it more suitable for our project.

Track Layout

Figure 1 below shows the layout of the track.

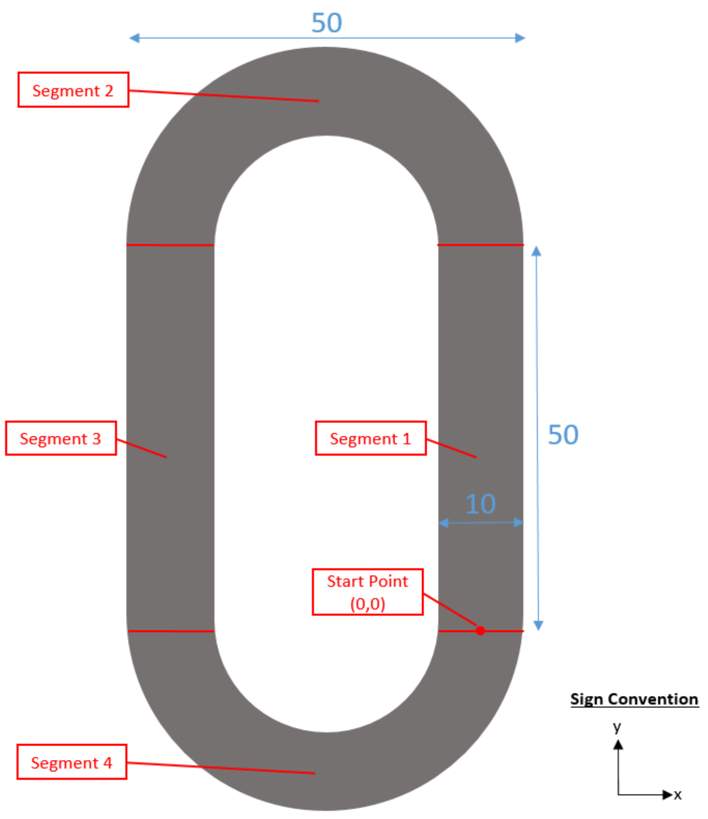

Figure 1: Track layout of the simulation

The track resembles a typical stadium track, and consists of 2 straight and 2 curved segments. The starting line is situated at the start of the first straight segment. The finish line is the same as the starting line. Initially, the track thickness was set to 4 metres. However, this was deemed too narrow for the car to travel along. The track was widened to 10 metres, which is acceptable as actual racing tracks are also reasonably wide. There are 3 checkpoints throughout the track along with the finish line, indicated as a red line on the track in Figure 1.

Training

At every episode, the car will be reset back to the starting point. The car will then continually take actions, which will be updated in the environment accordingly. As the car moves, a red trail is left behind for visualisation purposes. The episode ends when the car runs into one of the following termination conditions:

- Car crosses the finish line (measured from car centre)
- Car goes out of track (measured from car centre)
- Car stops moving (speed < 0)
- Car exceeds time limit (time elapsed > 500s)

The training was conducted for over more than 10,000 episodes in total. The model learned was saved every 150 episodes, and also whenever a new high score was reached for the average of past 100 scores. This was useful to restore previous models if they performed better than the latest model.

Results

Results of Initial Progress

The agent was trained a few times, with improvements made before the next training. The first training was done on a thin track that was 4 metres wide. From observation, it seemed that the agent was unable to navigate through the curved segment well. As this observation was made at an early episode count, there were 2 possibilities. The first possibility was that the track was too thin for the agent to navigate through. The second possibility was that the agent needed many more episodes to train. However, as it was too time consuming to wait and find out which of the 2 possibilities was true, the first possibility was assumed to be true. As a result, the track was made wider for the next training at 10 metre track thickness.

Before the next training, 3 checkpoints were also added. A new training was started and the agent was now able to go further than the previous training. Eventually, the agent was able to reach the finish line. However, the training time taken to achieve this was long. While this training was going on, additional rewards based on new distance travelled were added to the reward structure. A new training was carried out in parallel with the previous training after the changes. The agent was able to train slightly faster and complete the track. From these 2 trainings, it was observed that the robot would slow down a lot after it was able to reach the finish line. The path taken by the agent also did not seem optimal from observations. This is probably due to the lack of a time reward or penalty that incentivizes the agent to complete the track in the shortest time possible. As a result, the reward system was further improved with a time penalty and a time limit was set.

Final Results

The agent was finally trained over more than 10,000 episodes based on the final reward structure and track. The agent was able to clear the track faster than previous training attempts, which could be due to the new time penalty that was added. However, from further observations, the model started to perform worse after 9000 episodes. Due to previous models being saved every 150 episodes, the final model for the agent was taken slightly before 9000 episodes. The submitted code contains that final model, which was also the model used for validation in the submitted video.

Figure 2 below shows the graph of average scores by episode number. The average scores are calculated by taking the average of the past 100 scores that the agent got from the last 100 episodes. The score is determined by the total cumulative reward that the agent receives for that particular episode. Figure 3 below shows a similar graph, with average scores shown by steps taken instead.

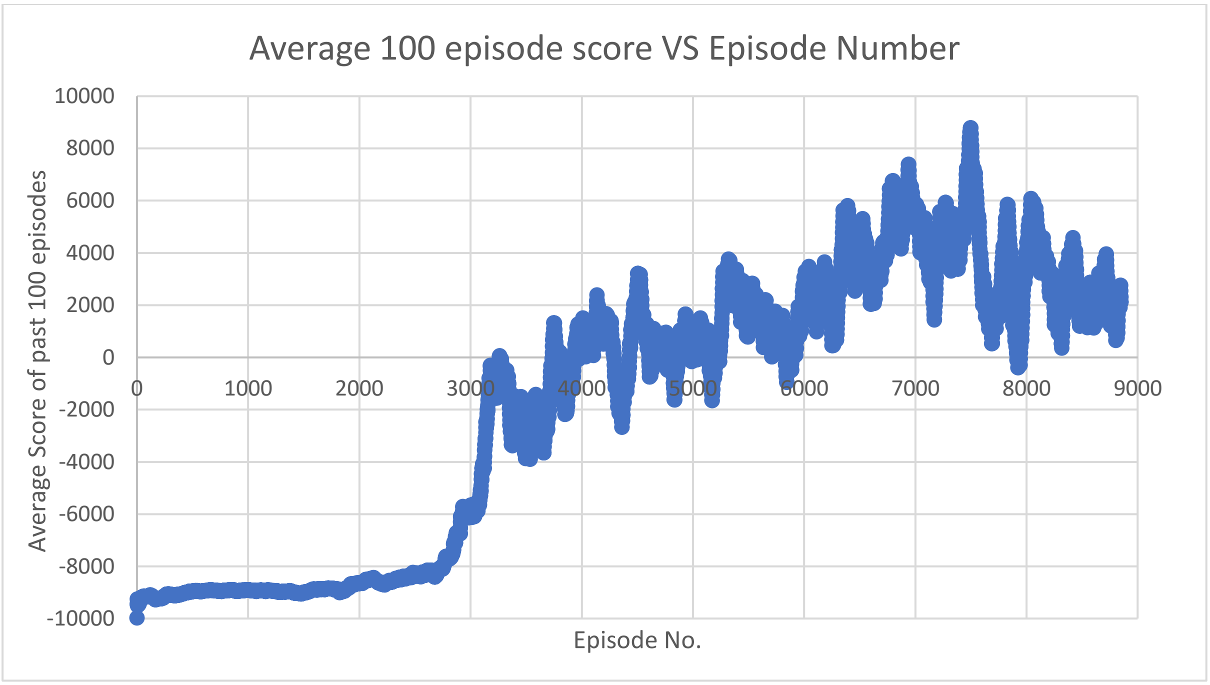

Figure 2: Graph of average scores by episode number

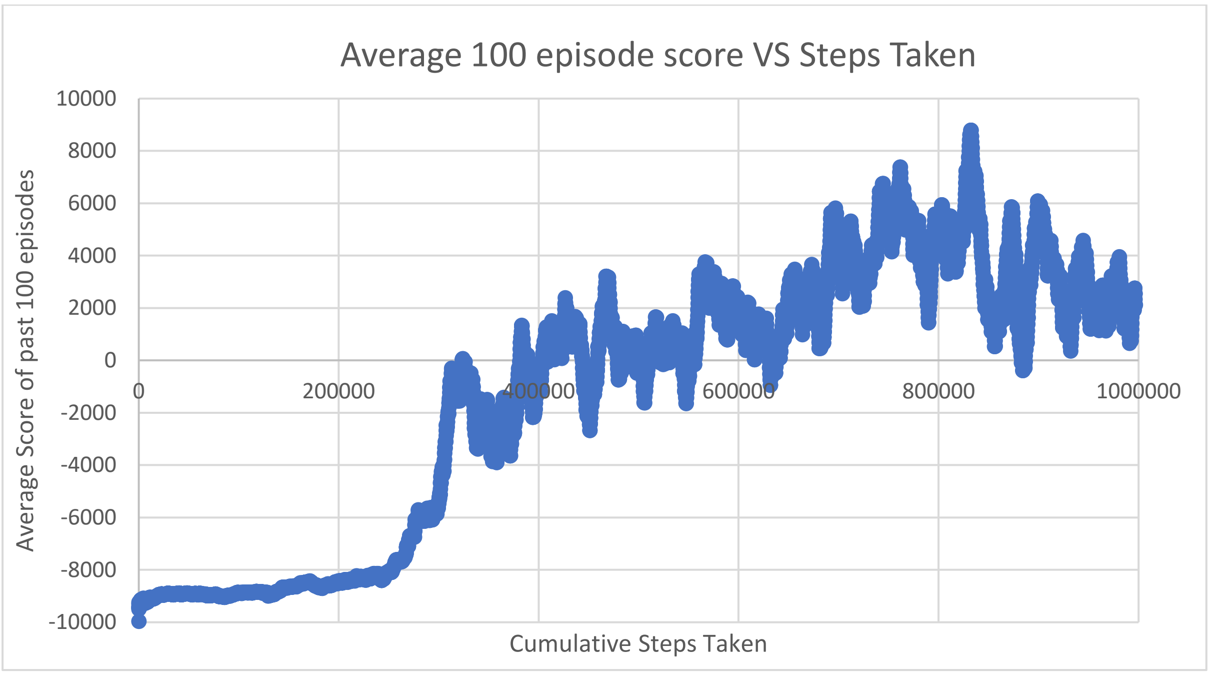

Figure 3: Graph of average scores by steps taken

Based on the 2 graphs, it can be seen that the scores generally improve as the training undergone by the agent increases. This result is expected as with more training data, the agent will be able to take better actions to maximise its rewards. However, after 7500 episodes of training, the scores start to drop. This could be due to the phenomenon of catastrophic interference, also known as catastrophic forgetting[3]. Catastrophic forgetting occurs in models that are trained sequentially. When new learning occurs, the model may adjust the weights in the neural network to fit the newer learning better. However, this occurs at the expense of the model previously representing older learning better. This loss of information results in the agent performing worse after it was already performing very well.

Optimal Path

As the agent started performing worse at the later episodes, the optimal performance was taken to be around 8000 to 9000 episodes. Therefore, the optimal path that the agent would take was gotten from a previously saved checkpoint around 9000 episodes. Figure 4 shows a rough trace of the final optimal path taken by the agent. The exact traced path can be seen from the validation in the video submission.

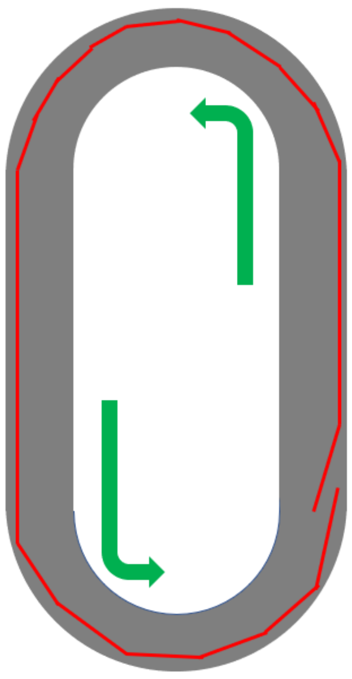

Figure 4: Final path taken by agent during validation

It can be seen that the agent stays close to the right side of the straight track segment and attempts the curve closer to the outer side. This is probably done to take wider turns over sharper turns, as sharper turns are harder to navigate at higher speeds. This is consistent with how race car drivers try to navigate at bends and corners[4].

However, this may not be the true optimal path that the agent can take to clock the fastest timings around the track. This could be due to the reward structure. Compared to the rewards such as 10,000 points for completion of track and 1,000 points for clearing checkpoints, the time penalty is only about 1 point for every additional second. The agent takes about 33 seconds to complete the optimal path shown in Figure 4. If the agent completed the track 5 seconds faster, it would only have 5 more additional points. As this is too small compared to its score of above 10,000, the agent may still prioritise higher rate of completion of track over time during training. To fix this, the time penalty could be made harsher so that the agent is sufficiently incentivised to clock faster timings.

Comparison to current methods

Current methods exist to solve this problem of finding the optimal racing line in the racing track. One such example is the computation of the racing line using Bayesian optimisation[5]. Another example is the Euler spiral method[6]. These methods heavily utilise mathematics and geometry to solve the problem. The advantages of these methods is that the computation can be done much more quickly than our RL approach, due to the time required to train an agent with RL. However, the mathematical approaches may not perform as well when the tracks become highly irregular. These methods may also need excessive amounts of information about the environment to accurately determine the optimal racing path. In contrast, our RL approach is able to work with lesser information to find the optimal racing path.

When comparing our methods to current methods, the current methods are better when there is a good amount of information available on the race track environment and there is limited time to solve the optimisation problem. Otherwise, our approach can be a good alternative to current methods to find optimal paths for any track with lesser information provided, given enough time.

Analysis of Proposed Methods

Our proposed method has many advantages and limitations. Some of it is due to our chosen approach, while others are due to the code implementation.

Advantages

One advantage of our method is our choice of using discrete action spaces. In reality, the steering, acceleration and braking actions are continuous in nature. However, this would drastically increase the training time required to train the agent. By using 5 discrete actions, the agent can train much faster, which is an important advantage under time constraints.

Another advantage of our method is the choice of time step of 0.1 seconds. It is possible to set much smaller timesteps to allow the agent to take more frequent actions. This may possibly find better optimal paths for the race track. However, as human reaction time lies between 0.2 to 0.3 seconds[7], human race car drivers will not realistically be able to take actions at a rate faster than 0.1 seconds. Therefore, choosing this time step is realistic, and also enables the robot to train faster than if a smaller time step was used.

Terminating the episode upon more conditions than just crossing the finish line enables the training time to be reduced as the agent trains faster. Currently, the episode terminates if the agent goes out of track, the agent tries to stop and move backwards, and the agent exceeds time limit. As these actions are undesirable and clearly do not lead to an optimal path, terminating episodes with such outcomes is reasonable.

Saving checkpoints during training is also an advantage of our method, as it allows us to pause and continue training from halfway. The checkpoints are also useful for going back to a previous learned model in the event it is better than the later model.

The images are also generated on a grayscale, and resized to make it smaller. This reduces the size of the input data, while keeping the quality of the input the same as the image generally has no details that can only be captured at a higher resolution. The line drawn to trace the path taken by the agent is also useful as it helps visualise how the optimal path on the race track looks like. This line symbolises the optimal racing line, and can be useful for race car drivers to refer to and learn the best path for the given track.

Limitations

There are some limitations with the approach that we took. The first limitation is the lack of a continuous action space. Although the use of a discrete action space was listed as an advantage, there is a trade-off based on the chosen action space. If a continuous action space was used, better optimal racing lines may have been found due to increased flexibility in actions. This however comes at a cost of greatly increased training time. To address this limitation, the discrete action space can be changed to a continuous action space, and more time can be allocated for training the agent.

Another limitation was the lack of friction and air resistance forces. In a physical environment, these forces can play a huge role in how the car moves based on the actions taken. Therefore, neglecting these forces may cause the simulation results to differ from the results in an actual environment. In our implementation, the forces were neglected to simplify the problem, due to the complexity of additionally adding in these forces to be computed in our custom environment. This limitation can be addressed by improving the simulation and adding the forces that the car will experience into the simulation environment.

The reward structure chosen also has its limitations in getting the agent to complete the track in the fastest possible time. As mentioned earlier, the time penalty given to the agent is too small. This results in time not being an important priority for the agent as rewards are the only indicator of desirable outcomes to the agent. To overcome this limitation, the time penalty can be increased to greater values. This may improve the performance of the agent on the track.

Challenges & Lessons Learned

Over the course of the project, there were many challenges that were encountered. These challenges were useful for learning new lessons from them.

Training Time

One such challenge is the unexpected length of training time required for the agent to train. When the training time is long, it is difficult to make changes to the code as that would require starting the training from scratch again, which is time consuming. As there is no way of knowing that the code has no mistakes when starting training, it is difficult to identify issues with the code until the behaviour of the agent is observed at later stages of the training. This results in less revisions and changes being able to be made to the code.

From this challenge, we learned many useful lessons on training. The first lesson was to ensure that we allocated more time for training. As it was our first Deep Reinforcement Learning project, we underestimated the training time. Another useful lesson learned is that we can have the training ongoing while we continue to make further changes to the code and start a second training in parallel to the first. This saves time and ensures that the first training is able to run longer so that more information about the training outcome can be obtained.

Lack of familiarity with RL

Another major challenge over the course of this project was the lack of familiarity with Reinforcement Learning. Although the concepts and theory are manageable, the code implementation is extremely challenging despite sufficient experience in programming. TensorFlow is a very useful library for writing machine learning programs. However, it is not easy to write neural networks using TensorFlow from scratch as there are many things to consider to ensure that the agent trains properly. As a result, we required help from online code to implement the agent and neural network portion of the code. Although the environment was coded from scratch, we had to learn Pygame and OpenGL to code the graphics part. Fortunately, these were more manageable to learn.

As we familiarised with the online code used, we were able to understand the implementation of neural networks and agents much better. This also enabled us to modify the code to suit the needs of our project. We also learned about the importance of checkpoints and their usefulness in restoring past models and resuming training from halfway points. Reinforcement learning also contains numerous algorithms and methods that can be implemented to solve the given problem. However, choosing the right algorithm is not easy as it requires a good amount of knowledge on them. Hence, we learned that we need to further expand our knowledge on the methods available out there.

Reward Structure

A good reward is necessary for the agent to be able to learn how to perform desired outcomes. As mentioned earlier, this is due to the reward being the only means of the agent getting feedback on which outcomes are more desirable than others. Hence, a good or bad reward structure can make or break the agent’s training outcome. However, we realised that the reward structure is not that easy to implement. Fortunately, the reward structure that we decided to go with in the end was sufficiently good to enable the agent to learn the track well and complete it in a relatively good timing.

Future Improvements

Based on what we have learned, these are some of the improvements that we can make to our problem.

Improved simulation

As the simulation used in our project is relatively simple, there are several improvements that can be made to improve it such that it resembles actual race conditions more closely. One such improvement is the addition of friction and air resistance forces. As mentioned earlier, this will simulate how the car will behave more accurately.

Another possible improvement is the use of a more complex track in the simulation. Although the track used in our simulation is simple, actual races contain tracks that have more bends which increase the complexity. On top of that, other cars can also be added to the simulation, as actual races also involve navigating amongst other racers. These other cars could be programmed to follow a certain predetermined optimal path. Alternatively, the other cars can also behave as agents which are trying to learn an optimal path through reinforcement learning at the same time.

Stochastic Problem

Another interesting improvement that may make the problem more realistic could be to make the problem a stochastic problem. Currently, the problem is a deterministic one where if the agent decides to take a certain action, that action will be taken 100% of the time. This may not be as useful for real race car drivers to learn from as there would probably be a small degree of variance in the exact action the driver wants to take, and the actual action that gets executed. This is due to the fact that it is unrealistic for drivers to have pinpoint accuracy. Hence, by modelling the problem as a stochastic one, there is a slight room for error, making the optimal racing line easier to follow.

Environmental conditions may also vary based on many factors such as weather, condition of the car and uneven friction along the whole track. Therefore, introducing probabilities may result in an optimal racing line that is more robust to changes in environmental conditions.

Other Improvements

As the reward structure has room for improvement, it can also be improved on in the future. In particular, the time penalty can be increased to a greater value to encourage the agent to clock faster timings. The action space can also be expanded from a discrete action space to a continuous action space to enable better possible outcomes by the agent. Currently, a DQN is used for solving the problem. However, it has certain issues such as catastrophic forgetting as mentioned above. This may impede the agent’s ability to learn longer race tracks that may need more training. Other algorithms may be able to alleviate this issue to enable the agent to train better.

References

[1]"How Do F1 Drivers Practice In The Off-Season? - One Stop Racing", *One Stop Racing*, 2021. [Online]. Available: https://onestopracing.com/how-do-f1-drivers-practice-in-the-off-season/. [Accessed: 19- Nov- 2021].

[2]"GitHub - jperod/AI-self-driving-race-car-Deep-Reinforcement-Learning: Solving OpenAI's reinforcement learning CarRacing environment", *GitHub*, 2021. [Online]. Available: https://github.com/jperod/AI-self-driving-race-car-Deep-Reinforcement-Learning. [Accessed: 19- Nov- 2021].

[3]"Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem", *Science Direct*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0079742108605368. [Accessed: 19- Nov- 2021].

[4]"How to Drive the Perfect Racing Line - Fast Through Apex & Exit", *Driver61*, 2021. [Online]. Available: https://driver61.com/uni/racing-line/. [Accessed: 19- Nov- 2021].

[5]"Computing the racing line using Bayesian optimization", *DeepAI*, 2021. [Online]. Available: https://deepai.org/publication/computing-the-racing-line-using-bayesian-optimization. [Accessed: 19- Nov- 2021].

[6]*Dspace.mit.edu*, 2021. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/64669/706825301-MIT.pdf. [Accessed: 19- Nov- 2021].

[7]"Human Benchmark", *Humanbenchmark.com*, 2021. [Online]. Available: https://humanbenchmark.com/tests/reactiontime. [Accessed: 19- Nov- 2021].