ISSN Number: 2454-2024(Online)

International Journal of Technical Research & Science

MACHINE LEARNING-BASED DYNAMIC TRAJECTORY OPTIMIZATION AND LAP TIME ESTIMATION FOR AUTONOMOUS RACING VEHICLES

Parth Patel, Madhusudan Barot, Krupal Shah
E-Mail Id: parthpatel31902@gmail.com, madhusudanbarot.me@indusuni.ac.in
Automobile Engineering, Indus Institute of Technology & Engineering, Ahmedabad, Gujarat, India

Abstract- The rapid advancement of autonomous vehicle technology has paved the way for autonomous racing, where the high-speed, competitive nature of motorsports fosters accelerated innovation. One of the primary challenges in this domain is determining the optimal path commonly referred to as the racing line for autonomous vehicles. Traditional methods for identifying such trajectories often either fall short in terms of time optimization or demand high computational resources, making them impractical for real-time execution on embedded hardware.

This paper presents a machine learning-based solution that enables real-time prediction of racing lines using desktop-level computing resources. The method utilizes a feedforward neural network trained on a dataset of racing lines derived from conventional optimal control-based lap time simulations across numerous circuits. The model achieves a mean absolute prediction error of ±0.27 meters, with an impressive ±0.11 meters accuracy at corner apexes comparable to both professional drivers and existing autonomous driving control systems. Notably, the system generates trajectory predictions in just 33 ms. These results highlight the potential of data-driven models to deliver near-optimal racing line predictions more efficiently than conventional computational methods, especially in time-sensitive applications.

Keywords: Self-Driving Race Car, Lap Time Prediction, Ideal Racing Trajectory, Path Planning, Artificial Neural Networks, Data-Driven Learning, Machine Learning.

1. INTRODUCTION

The rapid growth of Autonomous Vehicle (AV) technology holds the promise of significantly reducing traffic-related fatalities, as human error accounts for the vast majority of roadway incidents [1]. This progress has also fueled the emergence of autonomous racing a domain where the competitive and controlled setting of a racetrack serves as a powerful platform for testing, refining, and accelerating innovation in AV systems [2]. In such scenarios, vehicles are often designed to operate without any human intervention, and in some cases, they may be completely undrivable by a human operator [3]. As a result, the autonomous system must independently perceive its surroundings and generate its own motion plan making accurate and efficient path computation a critical component of its performance.

In motorsports, Lap Time Simulators (LTS) are widely utilized as a cost-efficient and practical means of analyzing how changes in vehicle configuration influence lap performance. These tools also facilitate optimization routines aimed at identifying the most effective setup for initiating on-track trials. However, due to variations in speed characteristics, corner geometry, and other track-specific factors, the optimal configuration derived for one circuit typically does not generalize well to others. Consequently, each racetrack necessitates its own dedicated optimization process.

Most Lap Time Simulators (LTS) employed by professional racing teams are built upon Quasi-Steady-State (QSS) methodologies [4]. In this framework, a vehicle model attempts to traverse a predefined path commonly segmented into a sequence of curves with varying radii at the maximum attainable velocity, typically achieved through iterative calculations. Some implementations further extend this approach to account for transient dynamic behavior of the vehicle [5]. However, the QSS technique generally relies on having prior knowledge of the desired path, which is often reconstructed from telemetry or trajectory data logged by expert drivers [6]. Yet, acquiring such data is both costly and challenging, even when assisted by GPS technology [7]. Moreover, this method assumes the driver followed the optimal line and that the vehicle can be physically driven on the track beforehand. Although it is possible to model a circuit using methods like aerial imagery [8], smaller racing teams often operating under budget constraints—may not have access to high-fidelity racing line data. In such cases, the available trajectory may be imprecise, noisy, or entirely absent.

In contrast, autonomous vehicles (AVs) must have the capability to compute their own driving trajectories preferably in real-time so they can adapt dynamically to changes in the environment, including road geometry and traffic conditions, without relying on extensive pre-processing. To drive innovation in this domain, many autonomous racing events incorporate trajectory planning challenges under uncertain or constrained conditions. These may include tracks that are initially unknown [9,10], disclosed only shortly before the race [11], mapped using limited sensing capabilities [12], or containing dynamic obstacles that alter the drivable space [13–16]. In

such scenarios, the AV may be allowed to construct a track map during an initial reconnaissance lap or adapt to evolving track layouts as it drives [17,18]. These events present considerable technical hurdles racing at high speeds necessitates extremely fast trajectory planning, while factors such as vehicle mass [19] and power limitations emphasize the need to execute this planning with minimal computational overhead. Furthermore, the computed path must not only be feasible but should also result in the shortest possible lap time.

Conventional fast methods for generating a target path often rely solely on the layout of the circuit, producing rough estimations that, while efficient, offer no assurance of achieving a time-optimal trajectory [18]. A significant number of current approaches to identifying the optimal racing line involve running extensive simulations across a wide range of potential paths using a vehicle dynamics model. These simulations then converge toward the trajectory that yields the lowest lap time. However, this optimization process is inherently computationally intensive and demands substantial processing power [20]. As a result, such methods can be prohibitively time-consuming when used in traditional Lap Time Simulators (LTS), making them impractical for real-time execution in autonomous vehicle systems.

Accelerating the generation of accurate trajectories to enable real-time execution would offer a substantial advantage for autonomous racing vehicles. Furthermore, quickly determining an optimal racing line for a given track could prove highly valuable not only in the context of traditional Lap Time Simulator (LTS) tools but also as an initial estimate to guide more precise, yet computationally intensive, optimization processes.

In various domains, model-based Machine Learning (ML) methods are frequently utilized to accelerate problem-solving by leveraging extensive training datasets to produce rapid predictions of the solution through a trained Artificial Neural Network (ANN). These techniques have been proven to address traditionally computationally intensive issues not only with speed but also with high accuracy in the outcomes [21].

The YOLO object detection algorithm, developed by Redmon and Farhadi [22], is widely used in autonomous racing vehicles. It significantly reduces the time required to process camera images to just a few milliseconds, enabling real-time, precise object recognition at high frame rates. This is achieved through a neural network trained on thousands of manually labeled images [23].

The Machine Learning (ML) approach to this type of problem focuses not on computationally solving the issue, but on making accurate predictions of the solution based on prior exposure to many similar problem scenarios through training. While neural networks have been used in vehicle control, there have been relatively few efforts to harness the speed and precision provided by ML techniques for the specific task of predicting an optimal racing line [24].

This paper presents an ANN approach to predict the ideal racing line, aimed at reducing the calculation time by several orders of magnitude. The network can be trained on data from any existing method of optimal trajectory generation, thus facilitating predictions based upon highly complex models to be made within milliseconds. This enables the target path to be calculated much more rapidly than with traditional methods reducing computational burden for traditional lap time simulators, and facilitating real-time application in an autonomous racing vehicle using on-board hardware. This original approach provides an unprecedented reduction in the time taken to generate the ideal racing line, with minimal loss in accuracy of the solution.

2. LITERATURE REVIEW

Basic methods for determining a target path typically involve following the shortest route or the Minimum Curvature Path (MCP), which Heilmeier et al. [25] demonstrated could be calculated in approximately 18 seconds for the Roborace autonomous racing vehicle. Although the MCP can be computed quickly, it only provides an approximation of the racing line – which may not lead to the fastest lap time [18] and does not account for the unique characteristics of different vehicles. When a higher level of precision in the racing line (and thus the lap time and vehicle setup) is required, this approach proves inadequate.

Most current methods for determining a time-optimal solution rely on the free-trajectory Optimal Control Problem (OCP) approach to lap time simulation. In this method, a simulator generates the necessary driver commands for a dynamic vehicle model to follow any path within the circuit's boundaries. The throttle, brake, and steering inputs are typically optimized at various discretized points on the track to achieve the minimum time, resulting in the optimal racing line [26]. This technique allows for the use of complex, fully dynamic vehicle models [27] and enables adjustments to the path based on changes in vehicle parameters [4], producing a highly accurate, vehicle-specific racing line.

However, due to the complexity of solving the vehicle model's trajectory at each discretized point, the computation time for the OCP approach is generally much longer than the actual lap time [20], with simulators that use simpler vehicle models taking around 15 minutes to solve a lap [7,28]. Lot and Dal Bianco's [29] more advanced model, with 14 Degrees of Freedom, took approximately 28 minutes to solve a lap. The free-trajectory method, essential for determining the racing line, usually requires significantly more time to solve compared to a LTS that follows a pre-defined path [30]. While OCP methods provide accurate, time-optimal solutions for the racing line, their computational demands typically make them unsuitable for real-time applications.

In an autonomous vehicle (AV), it is essential to perform these calculations in real-time or even faster, using onboard processing hardware. Jain and Moraro [31] tackled this challenge with Bayesian optimization, enabling

the calculation of a racing line in under three minutes. While this allows for quick pre-computation of a new circuit after an initial sighting lap, it is still not fast enough to compute the racing line for subsequent laps while driving the current one at speed. Christ et al. [32] modeled variable tire-road friction on the Berlin Formula E circuit for the Roborace autonomous vehicle, utilizing CasADi [33] to reduce the simulation time to under two minutes yet still slower than the lap time. Kapania et al. [18] achieved near real-time path planning through an iterative, two-step process (using the MCP as a preliminary calculation), generating a trajectory for an autonomous racing car within 26 seconds, which is faster than the lap time. However, the vehicle behavior was simplified to a bicycle model, leading to an approximation of the racing line that, while generally accurate, shows significant discrepancies of several meters in certain areas when compared to other trajectories [18].

Machine learning has been explored in the past for real-time vehicle control, often leading to the identification of an optimal path as a result of the vehicle's motion. For instance, Salem et al. [34] combined evolutionary learning with a fuzzy-logic controller to refine the vehicle's control inputs and racing line. This approach resulted in a controller capable of driving the vehicle around a circuit based on insights gained from previous iterative trials, although further development is needed for the vehicle to navigate multiple circuits. Yu et al. [35] applied Reinforcement Learning (RL) to learn basic control inputs for a simple 1980s Outrun racing game [36], while Balaji et al. [37] implemented an end-to-end RL system in a simulator to teach a real (1/18th scale) vehicle to drive using camera images. However, the vehicle was incentivized to follow the track's centerline rather than a true racing trajectory.

Another machine learning approach involves learning from human drivers. Fridman et al. [38] trained an end-to-end autonomous driving algorithm using 4.2 million video frames collected from a Tesla. However, most methods for road applications focus on safely navigating the road environment, rather than optimizing the path for speed. An end-to-end approach was tested for autonomous racing by Koppula [39], who observed that the method was "insufficient to produce smooth steering behaviors" in simulation. In general, end-to-end approaches tend to generate vehicle control inputs from sensor or camera data [40], rather than directly predicting a target path ahead.

There are few studies that directly apply machine learning to the task of generating a racing line. Cardamone [41] used a genetic evolution-based ML technique to iteratively refine the target path, ultimately finding the optimal trajectory for a specific circuit. Although the vehicle model used in this study was simple, limiting the accuracy of the generated line, it still represented a significant improvement over the MCP. A similar approach was adopted by Vesel [42], who introduced a 'healing' sub-process to enhance performance. However, these methods are restricted to providing a trajectory for the specific circuit on which they were trained. Weiss & Behl [43] utilized a convolutional neural network to learn a racing trajectory ahead of the vehicle in a racing game, generating vehicle control inputs that outperformed traditional end-to-end methods. However, it remains unclear if the system can handle a previously unseen circuit. In a similar vein, Capo & Loiacono [24] employed reinforcement learning to plan a short-term trajectory directly ahead of the vehicle for gaming applications. While this method attempts to minimize lap time and can plan a trajectory for a new circuit, the "learned behavior is still far from the performance of professional racing drivers" [24], partly due to the representation of the target path as a single point ahead of the vehicle.

At present, there is no widely accepted conventional method for generating a highly accurate racing line for a previously unseen circuit in real-time or faster. Machine learning techniques hold the potential to drastically reduce solution times, making their application to the task of identifying the ideal racing line particularly promising. This could enable the planning of a racing line at remarkable speed with minimal loss of accuracy, thereby supporting the real-time demands of an autonomous racing vehicle.

3. METHODOLOGY

This study utilizes a feed-forward artificial neural network (ANN) trained on a dataset of racing lines from various circuits. This allows for the rapid generation of racing lines for previously unseen circuits, based on the trajectories found in the training data. For an ANN to make accurate predictions, it must be trained on a dataset that covers the full spectrum of features that may be present in the new, unseen problem. As a result, an extensive dataset was created. The inner and outer boundaries of a large number of real circuits were reconstructed and then augmented to expand the dataset to include over 6,000 tracks, divided into 2.7 million individual segments. An optimal racing line for each circuit was generated using an existing OCP-based lap time simulation, thereby completing the dataset.

Each circuit is divided into smaller segments using the sliding window technique, allowing the network to learn the racing line for individual sections of the track rather than the entire circuit. This approach enables the network to handle circuits of varying lengths and generalize track sections across multiple circuits. After segmentation, data describing the vehicle's position on the track and the surrounding circuit geometry were extracted for various waypoints throughout the lap. These "features" were then used to train the network using the majority (86%) of the dataset.

Once trained, the ANN can calculate the features for a new, previously unseen circuit, thereby generating a prediction of the racing line. The network's hyperparameters were fine-tuned to enhance prediction accuracy, using an additional 9% of the dataset. K-Fold analysis [44] was employed for this process. Finally, the system's overall

performance is assessed by comparing the generated racing lines to the remaining 5% of the dataset, which consists of circuits the network has not encountered before. The methodology is detailed in the following subsections, with results presented in Section 3.

3.1 Training Data Generation

The dataset utilized in this study includes the inner and outer track boundaries, as well as the optimal racing line for each circuit. No specific method is prescribed for generating the ideal racing line in the dataset it could be derived from a high-order free-trajectory OCP method (e.g., Lot and Dal Bianco [29]), obtained through a Driver-in-Loop simulator (where it would reflect the preferred lines of a particular driver), or produced by any other conventional method for determining the racing line.

Before generating the optimal racing lines in the training data, it was first necessary to reconstruct a large set of circuits. The inner and outer boundaries were obtained for 82 real circuits from around the world, including several Formula 1 tracks (60 manually created and 22 sourced directly from TUMFTM [45]). In line with common practices in machine learning studies [23], augmentation techniques were applied to expand the dataset to a total of 6,058 circuits by scaling, flipping, and reversing the direction of travel for each track. Efforts were made to ensure that both the training and validation datasets contained an equal proportion of real circuits, with a larger share of real circuits reserved for final testing. A total of 288 circuits (comprising 9 real-world circuits and 279 augmented versions) were used as test data to evaluate the network's performance on a completely new, unseen circuit.

To generate the optimal trajectory for each circuit in the dataset, the Global Race Trajectory Optimization tool [46], developed by Christ et al. [32], was chosen. The default solver settings and vehicle parameters for the Roborace vehicle were used. This tool utilizes a double-track vehicle model with QSS weight transfer and non-linear tire models, which are converted into a nonlinear programming problem (NLP) using Gauss-Legendre collocation and solved with IPOPT. To reduce computation times, a curvilinear abscissa track description and the CasADi framework were employed. This approach strikes a balance between the accuracy of the generated racing line and the time required to compute optimal paths for each circuit in the dataset. Solving over 6,000 optimal laps would have taken approximately two weeks on a single processor thread, so to accelerate the process, the workload was distributed across multiple PCs with multi-core processors.

The training data was further expanded by dividing each circuit into a series of overlapping segments (or windows), resulting in a dataset containing over 2.7 million segments from different racetracks, each paired with the optimal racing line for that segment. The rationale, process, and benefits of this segmentation approach are detailed in Section 2.2. The total size of the dataset needed for accurate prediction of the ideal racing line was determined experimentally, following a methodology similar to that of Roh et al. [47].

All optimal trajectories were calculated under the assumption of a zero vehicle width, meaning the vehicle's centerline occupies the full width of the track. This approach allows for the exclusion of the vehicle width parameter from the training data, with the width being accounted for only when the network predicts the target path. As a result, the vehicle width can be adjusted after the network has been trained.

3.2 Circuit Sectoring Using Dynamic Windows

Each circuit was divided using a series of lines (or Normals) perpendicular to the track centerline, with the point where the vehicle trajectory intersects each line identified as a "waypoint." Although it is common [26,48] to vary the spacing between these lines to reduce resolution on straights (and thus reduce computational effort), a fixed interval of five meters was chosen. Maintaining a consistent interval benefits the network by removing an additional variable, thereby reducing the dimensionality of the input. In cases where two or more Normal lines intersect (e.g., at a tight hairpin bend), the lines are adjusted to become "Pseudo-Normals," where the angle between the Normal and the track centerline is modified away from 90 degrees until the lines no longer intersect (see Figure 1). This information is then encoded and fed into the network as detailed in Section 2.3.

Similar to how a human driver looks ahead on the circuit, several other studies [26,43] use a forward-looking "preview" to guide the vehicle's subsequent behavior. However, the racing line at any given point on the circuit is influenced by both the preceding curvature (which determines the vehicle's current position) and the upcoming curvature of the track. This allows the network to learn not only how the vehicle reached its current position but also where the trajectory is headed next.

Instead of passing all the track Normals for an entire circuit to the neural network, the data is divided into sections using the sliding window technique. This means the circuit is represented as a series of overlapping windows, each consisting of a fixed number of Normal lines and corresponding waypoints. This approach offers several advantages: the network input remains a fixed size, allowing it to handle circuits of varying lengths (e.g., Nürburgring Grand Prix circuit (5.14 km) and Red Bull Ring Circuit (4.31 km)); the network benefits from learning from a larger number of windows across different racetracks, helping it generalize behavior across circuits with similar features; and the sliding window method significantly increases the amount of data generated from each circuit, reducing the number of circuits needed in the training dataset.

Mathematically, each circuit is represented by a series of overlapping windows, each containing information about the track Normals both before and after the central Normal within the window. For a given Foresight value f, the

window (t_i) is formed by including the ordered set of track Normals surrounding and including the Normal line N_i ($N_{i-f,\ldots}, N_i, N_{i+f)}$. The neural network g is then trained on these windows to predict the location of the waypoints for a given track section, $g(t_i) \rightarrow w_i$, where w_i denotes the waypoint location on the Normal N_i. The tuning of the Foresight parameter is discussed in section 2.4.2. Once trained, the trajectory for an unseen track is calculated by predicting the waypoints for each window of the new circuit using the fully trained network.

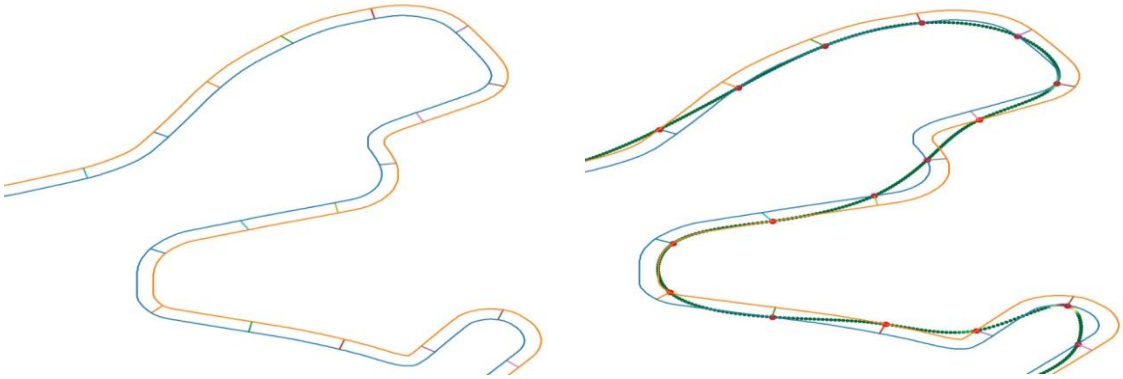

Fig. 3.1 (a) The circuit is segmented using lines that are perpendicular to the centreline of the track. (b) The racing line intersects with these segments at specific points known as waypoints

In this study, the Foresight is symmetric and optimized to minimize the prediction error for a circuit presented to the network in its entirety. However, the windowed approach means the trajectory is repeatedly planned for the area immediately surrounding the vehicle, which suggests that the Foresight could be linked to the perception sensors of an autonomous vehicle (AV). This makes the technique well-suited for rapid real-time trajectory planning in an autonomous road-going vehicle.

3.3 Extraction of Features

Instead of learning the racing line for an entire circuit, the network is trained on data describing how the circuit and its corresponding racing line flow through each of the windows outlined in section 2.2. This approach provides the ANN with the necessary data to infer key semantic information, allowing it to generalize a function that generates the target trajectory. To ensure that this information about the circuit and racing line is captured accurately, it must be encoded in a lossless manner. As such, features are extracted from the training data, including the circuit width (or length of each Normal), the racing line position on the track (or waypoint position along the Normal), and details about the curvature of the circuit at each Normal.

Many existing LTS represent the target racing line as a curvature profile, which helps in calculating maximum cornering velocities [6,49]. However, as the circuit curves, the radius of curvature changes in a highly nonlinear fashion, shrinking to just a few meters at tight hairpins and approaching infinity on straight sections. To address this, the circuit is instead represented by the variations between the Normal lines that describe its structure, rather than relying on the curvature itself (Figure 3.2).

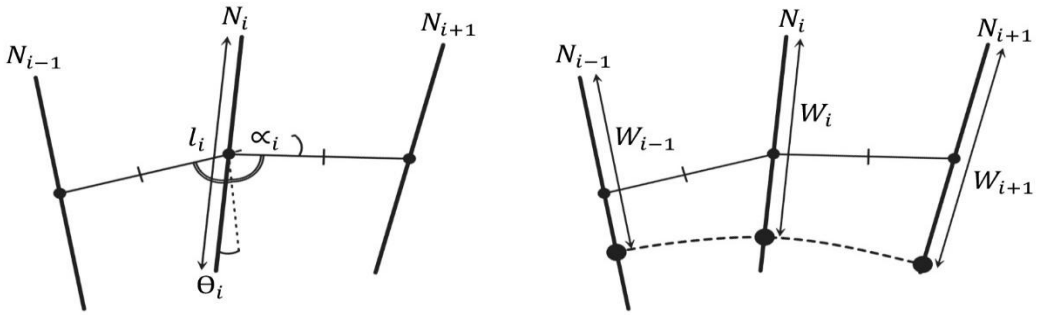

Fig. 3.2 The layout of the track is depicted on the left, while the right side illustrates the racing line using defined waypoints

The circuit geometry is represented by encoding the features for each Normal as follows: the length of the Normal l, the angular change between Normal α, and the angle θ between the Normal and the true normal to the track centerline (which is 90 degrees, except in the case of an adjusted 'pseudo-Normal'). These features are incorporated into the network by allocating three dimensions in the input space for each Normal within a given window (Figure 3). By preserving the sign of each angle, the system can differentiate between left and right-hand bends.

The racing line is characterized by determining the location of the waypoint along each Normal, denoted by the feature w, which ranges from zero to one, representing the left and right sides of the Normal, respectively. The

network is then trained using the w values for each window in the training data, allowing it to predict the racing trajectory for a new, unseen circuit.

3.4 Neural Network

A feed-forward ANN was chosen for this study due to the regression nature of the problem [50] and the relatively low computation time required for generating predictions with this type of network. This section provides an overview of the network's structure and the tuning process used to identify the optimal parameters, implemented using the Keras framework [51].

3.4.1 Architecture of the Neural Network

The network architecture consists of four fully connected layers, utilizing the Huber loss function [52] and the Nadam optimizer [53], chosen based on experiments with various hyperparameters. The number of units in each layer was determined through a grid search method, resulting in 250 units in the first layer, 50 in the second, and 50 in the third hidden layer (Figure 3).

The network performed optimally when the activation functions sigmoid and hard sigmoid [54] were applied to the hidden and output layers, respectively.

The output dimensions are defined by the sampling level s (as described in section 2.4.2). Training the network took about 90 minutes on a 2.40 GHz Intel(R) Core (TM) i5-9300H processor.

3.4.2 Sampling of Output

Reducing the output size of the network helps minimize dimensionality, which in turn enhances the network's ability to generalize and improves prediction accuracy. Instead of predicting every waypoint within a window, fewer waypoints are predicted and combined with those sampled from adjacent windows, thus constructing the complete predicted racing line for the entire lap.

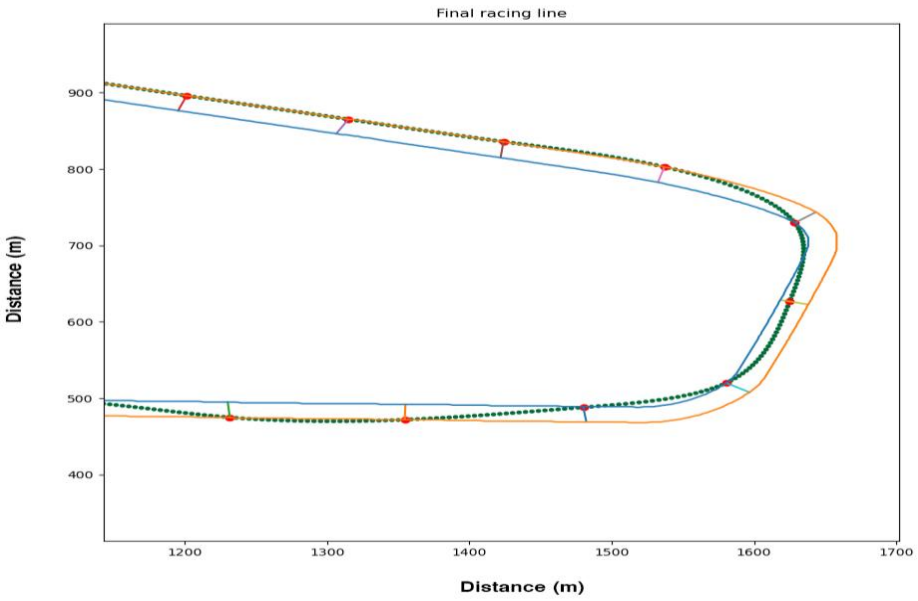

Fig. 3.3 Modifying the Sampling Level Results in a Smoothing Effect on the Predicted Trajectory

To ensure the racing line flows smoothly from one waypoint to the next, the network predicts multiple waypoints on either side of the central Normal within each window. These predicted waypoints are then averaged with those from neighboring windows, rather than relying on the single central waypoint in each window.

This approach creates a "moving average" effect for each predicted waypoint, resulting in a smoother racing line. The sampling level (s) controls how many pairs of surrounding waypoints the network predicts for each track segment. When $s = 0$, the network only predicts a single waypoint for each Normal, resulting in no averaging. However, when $s = 2$, the network predicts five waypoints (the central one and two on each side) for each Normal. After passing each track segment through the network, there will be $2s + 1$ predictions of the waypoint location for each Normal. These predictions are then averaged to produce a smoother racing line. This averaging process helps eliminate noise and ensures that the trajectory follows a more consistent and continuous path.

A sampling level of $s = 4$, which results in nine network outputs, was found to be the optimal choice for this study. This configuration provided the highest accuracy in predicting the racing line for the validation data.

By predicting multiple waypoints within each window and averaging them, the resulting racing line flowed smoothly from waypoint to waypoint. This approach eliminated the need for computationally expensive filtering techniques, such as those used by Heilmeier et al. [25], and ensured a more efficient and accurate prediction of the racing line.

4. RESULTS

To evaluate the performance of the ANN, predictions were made for circuits that were not part of the training data (i.e., previously unseen circuits in the testing dataset). These predictions were then compared against the racing lines generated using the Optimal Control Problem (OCP) method provided by TUMFTM [46]. The comparison focused on two main aspects:

- Accuracy of the Generated Line: The accuracy of the racing line predicted by the ANN was assessed by comparing it to the optimal line obtained through the OCP method. This evaluation helped determine how closely the ANN's predictions matched the ideal racing lines for each circuit.
- Solution Time: The time taken by the ANN to generate a racing line was compared to the computational time required by the OCP method. The goal was to demonstrate that the ANN could generate the racing line more rapidly, making it suitable for real-time applications in autonomous racing vehicles.

4.1 Racing Line Prediction Accuracy

The racing lines predicted by the ANN and the OCP method for the Brands Hatch GP and Nürburgring GP circuits are displayed below, with lateral deviation plots included for detailed comparison. Figure 4.1 presents the predicted racing lines for the Nürburgring Grand Prix circuit, comparing the ANN's prediction with the OCP-generated racing line. Similarly, Figure 6 shows the predicted racing lines for the Red Bull Ring Circuit.
These figures facilitate a visual and numerical comparison of the prediction accuracy, focusing on how closely each method follows the optimal racing line and the magnitude of lateral deviations at various points on the track.

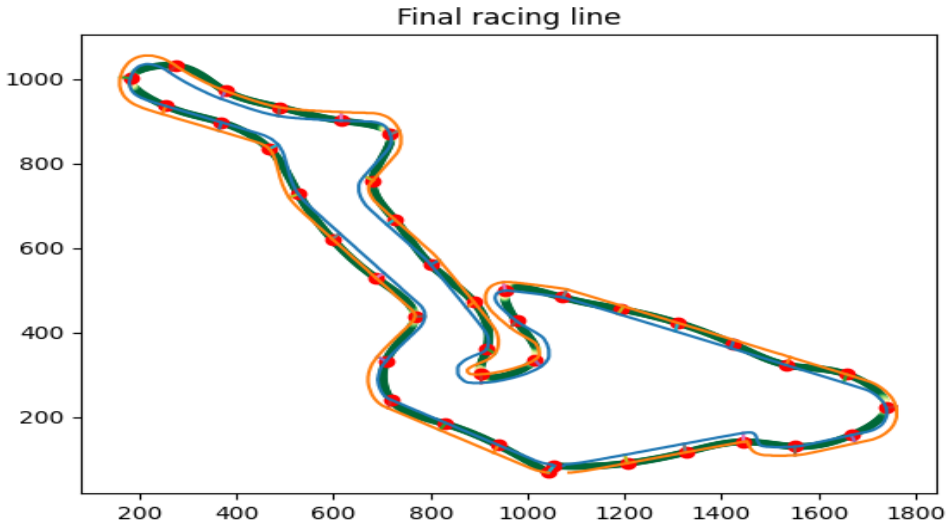

Fig. 4.1 A comparison between the racing line generated by the Artificial Neural Network (ANN) and the one derived using the Optimal Control Problem (OCP) approach for the Nürburgring Grand Prix circuit

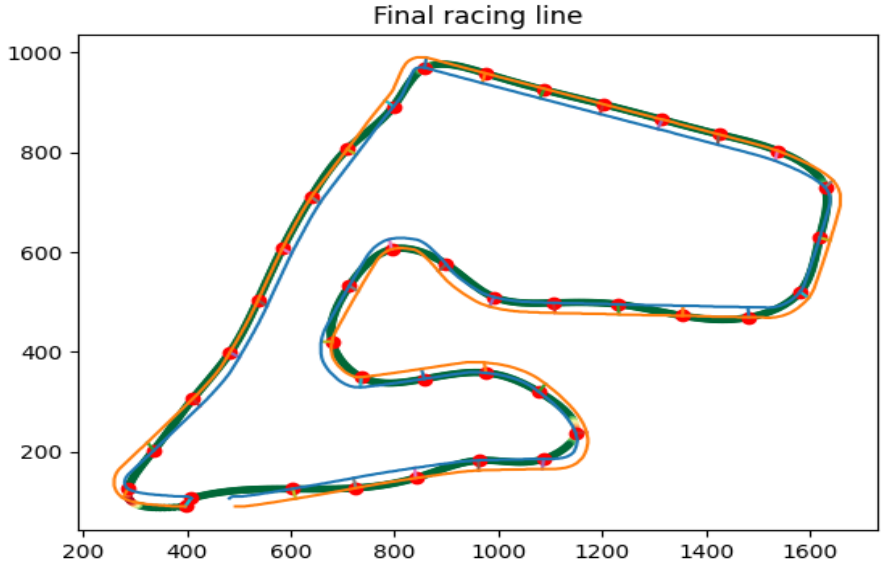

Fig. 4.2 A comparison between the racing line generated by the Artificial Neural Network (ANN) and the one derived using the Optimal Control Problem (OCP) approach for the Red Bull Ring Circuit

The trajectories predicted by the ANN are qualitatively similar to those calculated using the OCP method, following near-identical paths at most apexes, where the racing line has the most impact on lap time, due to the relative consistency of the optimal mid-corner racing line in the training data.

The maximum deviations between the ANN's prediction and the OCP method typically occur in areas with more complex and unusual features, such as the straight with a subtle bend before an upcoming corner at Nürburgring GP, or the intricate combination of consecutive turns that influence each other (Figure 4.1). Larger, sustained deviations are generally observed during straights; however, this behavior is common in many methods for calculating the optimal line and is considered less important, as a slight difference in vehicle position during a straight has minimal impact on lap time.

The highest accuracy is observed for different circuits, which feature characteristics frequently encountered in the training dataset, such as constant radius turns at the end of a straight, moderate-length straights, and average track widths.

Circuits that deviate significantly from the 'average' features in the training dataset such as those with complex corners of varying radii, rapid successions of multiple bends, or very short straights may present "edge cases" that are underrepresented in the training data or unseen by the ANN. For these circuits, the predicted racing line becomes a more generalized approximation biased toward the center of the dataset, leading to reduced prediction accuracy. This is evident in circuits like Paul Ricard (which features long straights and double-apex corners with varying radii) and the 20.8 km Nordschleife (which contains numerous complex corners and an exceptionally long straight). To mitigate the error caused by this sample bias, the training dataset should be centered around circuits with similar features to those for which predictions are needed. For instance, if predictions are required for a small kart track, training the ANN on a dataset consisting only of F1-style circuits would result in reduced accuracy. Diversification should be achieved by including a broader selection of original circuits in the training data, rather than relying on augmentations of a limited number of circuits as done in this study. Revealing a Mean Absolute Error (MAE) of ±0.27 m (±0.38 m Root Mean Squared Error (RMSE)) on average for all circuits in the testing dataset, reflecting the behavior observed across the nine real-world circuits.

The distribution of errors closely follows a Laplace distribution, with a 38% Confidence Interval (CI) of just ±0.18 m and a mean close to zero. The authors suggest that the symmetrical distribution with an almost zero offset is a result of flipping and reversing each circuit in the training data, ensuring the network was trained on an equal number of left and right-hand bends. At the corner apex where prediction accuracy is most critical the average error is reduced to just ±0.11 m across all circuits in the testing dataset.

The error distribution provides a 68% Confidence Interval (CI) of ±0.83 m, which is smaller than the variations observed in human racing drivers. For example, Brayshaw and Harrison [7] found that the lines taken by four professional racing drivers varied by more than ±1 m. This suggests that the ANN is capable of generating a trajectory comparable to a human racing driver's ability to identify and follow the optimal path, although many other factors can influence a human driver's choice of line. In real-world context, the prediction errors are smaller than the ±0.31 m MAE change in the optimal line due to localized (±10%) variations in the friction coefficient of the track surface observed by Christ et al. [32], or the ±1 m (95% CI) change in the optimal path resulting from adjustments in vehicle design parameters [7].

In comparison to existing methods for generating a racing line, the difference between the ANN's prediction and the OCP is of a similar magnitude to the variance observed between trajectories calculated by other traditional approaches. For instance, Dal Bianco et al. [49] observed an approximate apex error of ±0.08 m when comparing their direct and indirect methods, while the discrepancy between the ANN and OCP in this study is ±0.11 m. Kapania et al. [18] reported a much larger discrepancy of 'several meters' between trajectories calculated by the rapid 2-step algorithm and a traditional solver, which is considerably greater than the ±0.83 m (95% CI) difference between the ANN and OCP in this study. However, it is important to note that many existing methods calculate the trajectory alongside other data (such as velocity trace, accelerations, and vehicle control inputs), which is currently beyond the scope of this study.

From the perspective of autonomous vehicle (AV) control, the ANN's prediction accuracy is comparable to the accuracy of map generation in an autonomous racing vehicle at racing speed, as reported by Andresen et al. [56] (±0.39 m vs. ±0.29 m RMSE, respectively). The accuracy is qualitatively similar to that achieved by a path-following driver model following a racing line at speed in an autonomous racing car [23,57], as well as by a road-going AV in Wang et al. [58].

4.2 Computational Time

The typical solution time for generating a racing line for a previously unseen track using the neural network is 33 milliseconds on a 2.40 GHz Intel Core i5-9300H processor over 950 times faster than the OCP method for the majority of normal-length circuits. An interesting feature of the ANN approach is that the windows can be calculated simultaneously, whereas conventional approaches proceed iteratively around the circuit. As a result, solution times for very long circuits, such as the 20.8 km Nordschleife, are barely increased compared to shorter tracks 500 ms for the ANN versus 86 minutes for the OCP method. This makes the ANN approximately 5000 times faster at generating a racing line for this particular circuit.

The method proposed in this study is approximately 800 times faster than Kapania et al.'s [18] two-step approach, which, to the authors' knowledge, was previously the fastest known method for generating a non-trivial racing line, while also delivering a significantly more accurate solution.

DISCUSSION

The qualitative and quantitative comparisons presented in Section 3.1 demonstrate that this radically different approach to finding the racing line results in accurate predictions, offering comparable or reduced error compared to other traditional methods for calculating the optimal trajectory, the ability of a human racing driver to identify and follow an optimal path, the effect of small changes in friction or vehicle parameters, and an autonomous vehicle's ability to map and track a target path. However, the real benefit of this approach to racing line generation lies not in the accuracy of the prediction, but in the speed and computational efficiency of obtaining an accurate solution.

The ANN delivers an incredibly rapid prediction of the racing line, representing a vast reduction in solution time compared to traditional methods, with minimal sacrifice in accuracy. The current implementation is capable of making predictions for a pre-defined vehicle on an unknown circuit. However, it does not account for differences in track or weather conditions or adjustments to vehicle parameters such factors would require a more extensive dataset that includes such effects. Since changes to racing lines due to differences in vehicle parameters are typically small [7], this method enables the generation of a generic racing line for use in QSS simulations (i.e., the majority of widely-used LTS software).

The technique is ideally suited for generating the target path in real-time, making it highly applicable for online trajectory planning in an AV with minimal computational resources. The 'windowed' nature of the approach ensures that the trajectory is simultaneously planned for all sections of the road ahead, enabling the adaptation of the technique to address dynamically changing road conditions. This could include recovery after an unpredictable event or adjustments in trajectory to avoid other road users [59]. While such a method would require extensive offline pre-training (e.g., using datasets from real or simulated road events), the parallel nature of online planning allows multiple paths to be predicted concurrently. As a result, the AV could generate a trajectory through all possible routes in real-time, opening up exciting possibilities for applications like overtaking maneuvers, unexpected event recovery, and accident-avoidance systems.

Due to the long computational times required by many of the more accurate time-optimal approaches for finding the optimal racing line, a simple MCP or QSS pre-calculation is often used as a starting point to expedite the process by constraining the problem [57]. The method presented in this paper could be coupled with a time-optimal simulator to provide an exceptionally fast and accurate initial approximation. This could then serve as a pre-calculation, allowing for further fine-tuning through additional optimizations, such as minimum-time calculations, to refine the solution.

CONCLUSIONS

This work has demonstrated the potential of machine learning techniques to generate an extremely rapid prediction of the optimal trajectory around a previously unseen circuit. A feed-forward ANN was selected, with Sigmoid and Hard Sigmoid activation functions (on the hidden intermediate and output layers, respectively) found to deliver the best performance.

This approach does not solve the problem per se, but rather it rapidly generates an accurate prediction of the solution based on prior training with numerous similar problems. To achieve this, the network was trained on thousands of circuits. Specifically, this study used track boundaries for 6 real circuits, which were augmented to create a total of 96 tracks through various transformations such as scaling, flipping, and reversing.

An existing Optimal Control method was used to generate a racing line for each track, and the circuits were split into sections using a sliding window approach. This resulted in a dataset containing a total of 2.7 million track segments.

The ANN is designed to accept training data from any existing method of generating the racing line, providing a prediction based on that data. This means that if a large number of circuits driven by a human driver in a DIL simulator, or generated using a complex and accurate method, were used, the ANN would effectively "learn to drive like a particular driver (or simulator) in a specific car." The calculation time is independent of the model complexity used for training, so if a highly complex, free-trajectory simulator were employed to generate the training data, the accuracy of the solution would improve without impacting prediction time.

Predictions of racing lines for previously unseen circuits were found to have an average mean absolute error of ±0.27 m, with the highest accuracy at the critical corner apex (±0.11 m). The accuracy of the generated line is comparable to, and in many cases better than, other traditional methods for obtaining the racing line. This prediction accuracy aligns closely with the capability of a professional racing driver to identify and follow the optimal line, as well as the path-following accuracy of an autonomous vehicle control system.

The network delivers the most accurate predictions when the training data contains a variety of circuits with similar features. To minimize prediction errors, the dataset should focus on circuits of comparable size and complexity. For instance, if a prediction is required for a small kart track, training the ANN on a dataset composed

solely of F1-style circuits would lead to higher prediction errors. Furthermore, to reduce sample bias, the training data should be diversified by including a large number of original circuits, rather than relying heavily on augmented versions of a few circuits, as was done in this study.

The ANN generates a prediction for a full-size racing circuit in approximately 33 ms, making this approach over 9,000 times faster than the rapid OCP method and around 800 times faster than the fastest non-trivial approach reported in the literature survey. However, it is important to note that the machine learning approach requires extensive training on existing data, and many other methods also generate a speed trace, which is not addressed in the current paper's scope.

The capability to rapidly and accurately predict the optimal racing line makes this method the most efficient currently known for planning a complete or partial circuit trajectory. As such, it lends itself well to a variety of applications, including generating target paths for conventional simulators like QSS, performing pre-computations to shorten processing time in free-trajectory simulators such as OCP, and enabling real-time racing line estimation in autonomous race cars. Additionally, the method holds potential for adaptation in advanced use cases like overtaking strategies or collision-avoidance path planning in self-driving road vehicles. This approach challenges conventional methods for trajectory optimization—where brute-force calculations were the norm—by demonstrating that in specific scenarios, a data-driven strategy can achieve equal or greater efficiency without compromising accuracy.

REFERENCES

[1] National Highway Traffic Safety Administration, US Department of Transportation. Critical reasons for crashes investigated in the national motor vehicle crash causation survey, Traffic Safety Facts, No. DOT HS 812 115; 2015.

[2] Skeete JP. The obscure link between motorsport and energy efficient, low-carbon innovation: evidence from the UK and European Union. J Clean Prod. 2019;214:674–684. doi:10.1016/j.jclepro.2019.01.048.

[3] Betz J, Wischnewski A, Heilmeier A, et al. What can we learn from autonomous level-5 motorsport? In: 9th International Munich Chassis Symposium 2018. Wiesbaden: Springer Vieweg; 2019. p. 123–146. doi:10.1007/978-3-658-22050-1_12.

[4] Dal Bianco N, Lot R, Gadola M. Minimum time optimal control simulation of a GP2 race car. Inst Mech Eng D J Automob Eng. 2018;232(9):1180–1195. doi:10.1177/0954407017728158.

[5] Völkl T, Muehlmeier M, Winner H. Extended steady state lap time simulation for analyzing transient vehicle behavior. SAE Int J Passeng Cars-Mech Syst. 2013;6(2013-01-0806):283–292. doi:10.4271/2013-01-0806.

[6] Colunga IF, Bradley A. Modelling of transient cornering and suspension dynamics, and investigation into the control strategies for an ideal driver in a lap time simulator. Inst Mech Eng D J Automob Eng. 2014;228(10):1185–1199. doi:10.1177/0954407014525362.

[7] Brayshaw DL, Harrison MF. A quasi steady state approach to race car lap simulation in order to understand the effects of racing line and centre of gravity location. Inst Mech Eng D J Automob Eng. 2005;219(6):725–739. doi:10.1243/095440705X11211.

[8] Salazar-Lozano M, Medina-Murua PMCA. Low-cost 3d modelling of a go-karting circuit using drone based photogrammetry. DYNA. 2021;96(1):73–78.

[9] NATCAR [Internet]. UC Davis NATCAR; 2021 [cited 2021 Sep 9]. Available from: https://ece.ucdavis.edu/natcar.

[10] White D. [Internet]. EV Grand Prix - 2020 Rulebook [cited 2021 Sep 9]; 2020. Available from: http://evgrandprix.org/forms/rulebook/2020%20Autonomous%20Rulebook.pdf.

[11] NXP [Internet]. Rules, 2021 Concept, Calls Schedule, Campus Race Application; 2020 [cited 2021 Sep 9]. Available from: https://community.nxp.com/t5/The-NXP-Cup-EMEA/Rules2021-Concept-Calls-Schedule-Campus-Race-Application/ta-p/1106773?attachment-id = 106 421.

[12] FSAE [Internet]. Formula SAE; 2021 [cited 2021 Sep 9]. Available from: https://www.fsaeonline.com/cdsweb/gen/DocumentResources.aspx.

[13] FormulaPi [Internet]. Rules | Formula Pi; 2018 [cited 2021 Sep 9]. Available from: https://www.formulapi.com/rules.

[14] Zilly J, Tani J, Considine B, et al. The AI driving olympics at neurips 2018. In: Escalera S, Herbrich R, editors. The NeurIPS'18 competition. The Springer series on challenges in machine learning. Cham: Springer; 2020. p. 37–68. https://doi.org/10.1007/978-3-030-29135-8_3

[15] DeepRacer [Internet]. AWS DeepRacer – the fastest way to get rolling with machine learning; 2021 [cited 2021 Sep 9]. https://aws.amazon.com/deepracer/

[16] F1Tenth [Internet]. F1TENTH IROS 2021 | F1TENTH Rules; 2021 [cited 2021 Sep 9]. Available from: https://iros2021.f1tenth.org/rules.html.

[17] Kabzan J, Valls MI, Reijgwart VJ, et al. Amz driverless: The full autonomous racing system. J Field Robot. 2020;37(7):1267–1294. [18] Kapania NR, Subosits J, Christian Gerdes J. A sequential two-step algorithm for fast generation of vehicle racing trajectories. J Dyn Syst Meas Control. 2016;138(9). doi:10.1115/1.4033311.

[18] Patil M, Johri A, Sharma V, et al. Analyzing the performance of a formula type race car using lap time simulation. Indian J Sci Technol. 2016;9(39). doi:10.17485/ijst/2016/v9i39/94146.

[19] Lenzo B, Rossi V. A simple mono-dimensional approach for lap time optimisation. Applied Sciences. 2020;10(4):1498), doi:10.3390/app10041498.

[20] AlQuraishi M. End-to-end differentiable learning of protein structure. Cell Syst. 2019;8(4):292– 301. doi:10.1016/j.cels.2019.03.006.

[21] Redmon J, Farhadi A. Yolov3: An incremental improvement; 2018. arXiv preprint arXiv:1804.02767.

[22] Culley J, Garlick S, Esteller EG, et al. System design for a driverless autonomous racing vehicle. Communication systems, networks and digital signal processing (CSNDSP). IEEE; 2020. p. 1–6. doi:10.1109/CSNDSP49049.2020.9249626.

[23] Capo E, Loiacono D. Short-term trajectory planning in TORCS using deep reinforcement learning. 2020 IEEE symposium series on computational intelligence (SSCI). IEEE; 2020. p. 2327–2334, December.

[24] Heilmeier A, Wischnewski A, Hermansdorfer L, et al. Minimum curvature trajectory planning and control for an autonomous race car. Veh Syst Dyn. 2019: 1–31. doi:10.1080/00423114.2019.1631455.

[25] Casanova D. On minimum time vehicle manoeuvring: The theoretical optimal lap [PhD thesis]. Cranfield University; 2000.
https://hdl.handle.net/1826/1091

[26] Kelly DP. Lap time simulation with transient vehicle and tyre dynamics [PhD thesis]. Cranfield University; 2008.
https://dspace.lib.cranfield.ac.uk/handle/1826/4791

[27] Perantoni G, Limebeer DJ. Optimal control for a formula one car with variable parameters. Veh Syst Dyn. 2014;52(5):653–678. doi:10.1080/00423114.2014.889315.

[28] Lot R, Dal Bianco N. The significance of high-order dynamics in lap time simulations. In: The dynamics of vehicles on roads and tracks: International Association for Vehicle System Dynamics (IAVSD 2015). 2015. p. 553–562. doi:10.1201/b21185-59.

[29] Veneri M, Massaro M. A free-trajectory quasi-steady-state optimal-control method for minimum-time problems of cars and motorcycles. In: Dynamics of vehicles on roads and tracks. Cham: Springer; 2019. p. 1264–1270. doi:10.1080/00423114.2019.1608364.

[30] Jain A, Morari M. Computing the racing line using Bayesian optimization; 2020. arXiv preprint arXiv:2002.04794.

[31] Christ F, Wischnewski A, Heilmeier A, et al. Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients. Veh Syst Dyn. 2019: 1–25. doi:10.1080/00423114.2019.1704804.

[32] Andersson JA, Gillis J, Horn G, et al. CasADi: A software framework for nonlinear optimization and optimal control. Math Program Comput. 2019;11(1):1–36. doi:10.5281/zenodo.1257968. [34] Salem M, Mora AM, Merelo JJ. Beating uncertainty in racing bot evolution through enhanced exploration and pole position selection. In: IEEE Conference on Games (CoG). IEEE; 2019. p. 1–8. doi:10.1109/CIG.2019.8847998.

[33] Yu A, Palefsky-Smith R, Bedi R. Deep reinforcement learning for simulated autonomous vehicle control. Course Project Reports: Winter; 2016.

[34] Suzuki Y. Outrun, classic arcade racing game; 1986.

[35] Balaji B, Mallya S, Genc S, et al. Deepracer: autonomous racing platform for experimentation with sim2real reinforcement learning. In: IEEE international conference on robotics and automation (ICRA). IEEE. 2020. p. 2746–2754. doi:10.1109/ICRA40945.2020.9197465.

[36] Fridman L, Ding L, Jenik B, et al. Arguing machines: human supervision of black box AI systems that make life-critical decisions. IEEE conference on computer vision and pattern recognition workshops; 2019; doi:10.1109/cvprw.2019.00173.

[37] Koppula S. Learning a CNN-based end-to-end controller for a formula SAE racecar; 2017. arXiv preprint arXiv:1708.02215.

[38] Tampuu A, Matiisen T, Semikin M, et al. A survey of end-to-end driving: architectures and training methods. IEEE Trans Neural Netw Learn Syst. 2020.
doi:10.1109/TNNLS.2020.3043 505.

[39] Cardamone L, Loiacono D, Lanzi PL, et al. Searching for the optimal racing line using genetic algorithms). IEEE conference on computational intelligence and games; 2010; doi:10.1109/ITW.2010.5593330.

[40] Vesel R. Racing line optimization@ race optimal. ACM SIGEVOlution. 2015;7(2-3):12–20. doi:10.1145/2815474.2815476.

[41] Weiss T, Behl M. DeepRacing: parameterized trajectories for autonomous racing; 2020. arXiv preprint arXiv:2005.05178.

[42] Stone M. Cross-validatory choice and assessment of statistical predictions. J R Stat Soc Series B Methodol. 1974;36(2):111–133. doi:10.1111/j.2517-6161.1974.tb00994.x
[44] TUMFTM [Internet] TUMFTM/global_racetrajectory_optimization, GitHub [cited 2021 Sep 9]; 2019. https://github.com/TUMFTM/global_racetrajectory_optimization.
[45] Roh Y, Heo G, Whang SE. A survey on data collection for machine learning: a big dataAI integration perspective. IEEE Trans Knowl Data Eng. 2019. doi:10.1109/TKDE.2019.2946 162.
[46] Botta M, Gautieri V, Loiacono D, et al. Evolving the optimal racing line in a high-end racing game. In: IEEE conference on computational intelligence and games (CIG). IEEE; 2012. p. 108–115. doi:10.1109/CIG.2012.6374145.
[47] Dal Bianco N, Bertolazzi E, Biral F, et al. Comparison of direct and indirect methods for minimum lap time optimal control problems. Veh Syst Dyn. 2019;57(5):665–696. doi:10.1080/00423114.2018.1480048.
[48] Lathuilière S, Mesejo P, Alameda-Pineda X, et al. A comprehensive analysis of deep regression. IEEE Trans Pattern Anal Mach Intell. 2019. doi:10.1109/TPAMI.2019.2910523.
[49] Chollet F, et al. [Internet]. Keras, GitHub; 2015 [cite 2021 Aug 29]. Available from: https://github.com/fchollet/keras.
[50] Huber PJ. Robust statistics. Vol. 523. John Wiley & Sons; 2004. doi:10.1002/0471725250.
[51] Dozat T. Incorporating Nesterov momentum into Adam. In: International conference on learning representations workshop; 2016. p. 1–4.
[52] Gulcehre C, Moczulski M, Denil M, et al. Noisy activation functions. International conference on machine learning. 2016. p. 3059–3068. arXiv preprint arXiv:1603.00391.
[53] Bauer B, Kohler M. On deep learning as a remedy for the curse of dimensionality in nonparametric regression. Ann Stat. 2019;47(4):2261–2285.
[54] Andresen L, Brandemuehl A, Hönger A, et al. Accurate mapping and planning for autonomous racing. In: IEEE/RSJ international conference on intelligent robots and systems (IROS). 2020. p. 4743–4749. arXiv preprint arXiv:2003.05266
[55] Kapania NR. Trajectory planning and control for an autonomous race vehicle. Stanford University; 2016. https://ddl.stanford.edu/publications/trajectory-planning-and-control-auto nomous-race-vehicle.
[56] Wang R, Jing H, Hu C, et al. Robust h∞ path following control for autonomous ground vehicles with delay and data dropout. IEEE Trans Intell Transp Syst. 2016;17(7):2042–2050. doi:10.1109/TITS.2015.2498157.
[57] Erlien SM, Fujita S, Gerdes JC. Safe driving envelopes for shared control of ground vehicles. IFAC Proceed Vol. 2013;46(21):831–836. doi:10.3182/20130904-4-JP-2042.00096.