POLITECNICO DI TORINO

MASTER's Degree in AUTOMOTIVE ENGINEERING

Politecnico di Torino

MASTER's Degree Thesis

Racing line optimisation algorithms for high performance and/or automated vehicles

Supervisors

Prof. ALESSANDRO VIGLIANI

Prof. ALDO SORNIOTTI

Ing. PIETRO STANO

Candidate

PAOLO RUSSO

OCTOBER 2023

Abstract

The lap time optimisation is the process to calculate the best trajectory to achieve the minimum lap time, while considering the boundaries due to the vehicle dynamics. Traditional approaches have leaned on physics-based control models, which, although effective in offline settings, fall short when applied to online racing scenarios. These models not only burden computational resources but also lack the adaptability required to respond promptly to dynamic variations in vehicle behaviour and track conditions.

The work of this thesis consists of a reinforcement learning algorithm capable to act as a path re-planner with access to previously stored information coming from previous manoeuvres. This introduces the way for adaptive re-planning that can dynamically respond to changes in the vehicle's behaviour and evolving track conditions during the course of a racing session. Such dynamic changes include variations in friction coefficients, alterations in tire temperature, and fluctuations in the dynamic condition of the vehicle—all of which are common occurrences during a racing session.

The primary objective of this study is to optimise lap times by considering real-time vehicle information, gathered through data collection by sensors such as accelerometers, yaw rate and wheels speeds from previous and current manoeuvres. To achieve this, two critical components are developed and integrated into the system: a highly representative 8-degree-of-freedom (DOF) vehicle model and a data buffer for storing historical performance data from prior laps.

The 8-DOF vehicle model serves as a comprehensive representation of vehicle dynamics, forming the foundation upon which the re-planning process operates. The trajectory derived from this process guides the vehicle's path in the current state through a previously tuned Feedforward-Feedback (FF-FB) path tracking controller.

Data extrapolation and retrieval rely on the use of buffers—databases storing dynamic information of the vehicle, including steering actions, lateral and longitudinal accelerations, as well as vehicle states such as position and yaw. This stored data is then accessed by the RL algorithm within a spatial window determined by the vehicle's current position. By focusing on localised data retrieval around the vehicle, the overall procedure of reading the database is facilitated, allowing for more efficient agent learning.

In summary, the novelty of this research lies in the creation of an architecture that enables online learning based on data recorded from previous manoeuvres. This approach promises to enhance the performance and safety of autonomous racing vehicles, ultimately contributing to the advancement of autonomous driving technology in high-speed and dynamic environments.

A Vittorio ed Angela. Il vostro supporto è stato tale che questo raggiungimento è più vostro, che mio.

Acknowledgements

Un sincero ringraziamento al professore Aldo Sorniotti per il lavoro dedicatomi e per le continue delucidazioni in ambito di dinamica del veicolo. Un ringraziamento sentito va a Pietro Stano, faro di riferimento, umanamente e tecnicamente, in questi mesi di lavoro. Un profondo ringraziamento anche ai colleghi dell'ufficio 13AA03, Giulio, Nuccio, Edoardo, Paolo, Matteo ed Ignazio; condividere questa esperienza con voi mi ha fatto sentire fortunato.

Vorrei esprimere la mia gratitudine anche a Federico, Davide, Jaime e Francesco; non potevo chiedere una compagnia migliore per sentirmi a mio agio ovunque fossi.

A Piero e Nina, la cui saggezza mi hanno sempre guidato e rasserenato nelle decisioni più difficili.

A tutti i colleghi, amici e conoscenti incontrati durante questi anni e sparsi per il globo; spero di essere riuscito nel tempo a dimostrare la mia profonda gratitudine verso ciascuno di voi.

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

MLTS

Minimum lap time simulation

OCP

Optimal control problem

NN

Neural Network

RL

Reinforcement Learning

DOF

Degrees Of Freedom

QSS

Quasi steady state

NLP

Non linear programming

Chapter 1

Introduction

In the automotive industry, racing has always been the benchmark for innovative technologies. In the context of racing, the parameter most commonly used as an index of performance is the amount of time required to cover a defined distance, referred to as the lap time [1] [2]. In order to achieve the best performance, the lap time must be as low as possible and, in the context of a race, the objective is to minimise consecutive lap times as much as possible, resulting in the lowest possible overall time. In recent years, thanks to the evolution of numerical analysis and computing power, the goal of minimising lap times has been handled through simulations, resulting in what is now known as Minimum Lap Time simulations (MLTS) [3]. These simulations address the challenge of minimising lap times by considering an optimal control problem (OCP). The core objective of the OCP is to control the vehicle's power delivery, deceleration, and driving techniques to achieve the best possible lap time. Essentially, MLTS represents an optimal vehicle control methodology focused on non-linear optimal control techniques.

1.1 The Minimum lap time simulation

Throughout history, the way in which minimum lap time simulation has been handled as an optimal control problem has been approached in different ways. First of all, it must be emphasised that MLTS was not always approached as an OCP. Instead, the first experiments focused on simulating already defined trajectories as

truthfully as possible, helping drivers and engineers to analyse the data without necessarily having to refer to an actual driven lap [4].

Only later, when the technology was mature enough to allow the required computational effort, people begin to consider MLTS as a tool for estimating the best possible trajectory, instead of just simulating a predefined one. Based on this idea, lap time was defined as a cost function to be minimised, subject to the model physics and physical limitations of the track, resulting in an optimised trajectory. In other words, the MLTS was interpreted as an OCP. [3].

Several approaches have been proposed over the years concerning not only how the OCP was imposed mathematically, but also on what the control objective was. In fact, there are several applications where the focus has been not so much on generating trajectories but trying to compute appropriate velocity and acceleration profiles in order to assets the lowest lap time achievable [5]. Or, again, treating the OCP as an actual calculation of driver inputs in order to perform the best lap time possible, extending the concept of MLTS to a calculation of virtual drivers capable of competing with physical expert drivers [6].

1.2 Aim of this thesis

Having explained the historical context in which the writing of this thesis takes place, the project presented is positioned as a research study of the possible application of a reinforcement learning (RL) agent to calculate the best possible trajectory given a circuit. During the learning process, the agent consults a buffer of data that enables it to improve its performance through prior experience knowledge. The novelty of this thesis is therefore to propose a tool-chain where an agent can learn by accumulating direct experience on a lap time simulator. In addition, for a more complete research, a second tool-chain was created in which the agent acts on a highly realistic vehicle model and, thanks to the implementation of specially created buffers, the agent is also able to improve himself thanks to information from previous laps.

At the best of the author's knowledge, the state of the art application of artificial intelligence on trajectory computation is limited to either the computation of control actions or, if trajectories are computed based on geometric information, they are not tested on simulators, thus demonstrating convergence to a solution of which there are no actual proof of its performance over a simulated lap time.

1.3 Chapter organisation

The chapter 1 has been just a brief introduction to the Minimum Lap Time (MLT) problem. In the following chapters a more in depth analysis of the work carried out will be exposed.
In chapter 2 a literature review has been proposed, with emphasis on the different approaches adopted during the years concerning MLT. Then a standard methodology has been proposed, which can be helpful for any MLT application since it can be adapted to any kind of approach. Finally a series of detailed tables resuming the state-of-art is presented in the last section of this chapter, putting in evidence the most adopted solutions up to date.
For what concern the chapter 3, an introduction to RL has been addressed with its definition and why it has been chosen for the development of this thesis.
Furthermore, the tool-chain used for this thesis is explained in the chapter 4, explaining all the steps involved in developing a proper lap time simulator in Simulink environment and how to couple it with a RL agent.
In chapter 5, the second tool-chain is presented, which group the RL with the 8-degree-of-freedom vehicle model, the buffer implementation and a brief explanation of the controllers used.
In chapter 6, on the other hand, the implementation of buffers is explained in more detail, starting with their theoretical explanation through to their formulation and implementation.
Finally, in the last chapter 7, it is possible to find the results obtained and the conclusion drawn form them. A critical analysis is performed, questioning if the goal set has been reached and highlighting the benefits and drawbacks of such applications in further studies.

Chapter 2

Literature review

The state of the art on MLTS is very broad and varied, being a problem that has been posed since the early days of automotive racing [3]. Different methodologies have been proposed over the years, and with them there are as many diversification of proposed solutions, considerate models and programs used. The aim of the following chapter is, therefore, to give guidelines on the state of the art, organising the state of the art into macro groups, each one discussed into a dedicated subsection:

- If the simulation has been carried out over a predetermined trajectory or it is computed by scratch (subsection 2.2.1).
- How the optimisation problem has been addresses, if the goal was to minimise the lap time or the path driven by the car (subsection 2.2.2).
- The vehicle models adopted, since they vary from one-point mass vehicle to highly complex models up to 7 DOF (subsection 2.2.3).

2.1 OCP: Fundamentals

Before delving into the different proposed sections, it is good to first define how the concept of OCP is generalised in the case of an MLTS, since it is a formulation that not only helps in the understanding of the different sections but, as a generalist writing, also allows it to be extended to any application of MLTS. The general

formulation of a OCP applied to a MLTS can be write in the following way [7]:

$$\begin{aligned} \text{minimise } & \int_0^T 1\, dt \\ \text{subjected to: } & \dot{x} = f_c(x(t), u(t)) \\ & x(0) = x_S, \quad x(T) = \mathcal{X}_F \\ & x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U} \end{aligned} \tag{2.1}$$

Where the lap time time is defined as T, all the controller action (human or algorithm) is defined by $u(t)$, the vehicle model is defined by the state equation $f_c(x, u)$, x_s and $x(T)$ represent respectively the initial and final conditions (starting and finish line), meanwhile $\mathcal{U}$ and $\mathcal{X}$ represent the actuation and track constrains.

From the generic formulation (2.1), it is possible to derive most of the OCPs applied to MLTS. This simple formulation is fundamental for the understanding of the following part, where a classification of the literature is given, where the groups identified mainly concern the argument of the minimum function, the vehicle model considered and the physical limits of the track.

2.2 MLTS: General approaches

When interfacing with the trajectory optimisation problem some engineering choices must be taken. The latter concerns the formulation of the minimisation problem, the vehicle model and if the optimisation calculate a trajectory from scratch or improve a reference one. However, these choices are linked among them, meaning that, for example, the use of a simplified vehicle model well suits a quasi-steady-state simulation due to their light but reliable lap time modelisation [3]. Three approaches established them-self during the years and they can be summarised as follows:

- **QSS with fixed trajectory** ('apex finding technique'): This technique is the oldest one and offer a significant advantage with its quick algorithms, requiring minimal computational effort for both the model and simulations. This methodology, as the name suggest, consider that the lap time is divided

into several steps and in each one of these the vehicle is considered to be in a steady state condition. This condition permit to properly define the vehicle state along with its handling limits thanks to the use of g-g diagrams, which are fundamental for this kind of applications.

The latter are plots of the maximum longitudinal acceleration of the vehicle for a specific speed versus the lateral acceleration obtained for the same speed. Their application starts from the early days of racing performance analysis since they permit to define the concept of handling limit. When a vehicle's steady state falls within the envelop of the friction ellipse on the g-g diagram, it signifies that the vehicle has reached its maximum combined acceleration. The causes for reaching this limit can vary, ranging from tire friction limits to engine power limits.

By considering acceleration data, both from simulations and recorded on-track measurements, and comparing them with the g-g diagrams it becomes possible to determine the margin to the handling limit. The gap between the data and the envelop on the g-g diagram represents the residual capacity to fully exploit the vehicle's handling capabilities [4].

Due to its simplicity in both modelling and application, in the literature it has been studied even applications of such method utilising RL agents to create command inputs to reach this handling limit [5], which can provide performance comparable to a skilled driver.

Furthermore, g-g diagrams can also be extended into 3D maps if the velocity is considered as the third variable. This extension becomes valuable in scenarios where the velocity profile needs to be calculated based on a predefined trajectory [3].

This methodology is more adequate to the vehicle performance analysis, as the presentation [8] shows. Instead of conducting real track tests for each setup change, simulations are performed using a lap time simulator, enabling the identification of optimal setups even before the actual testing begins. This becomes particularly crucial in categories like the Formula SAE competition, where time reserved for testing is limited [9].

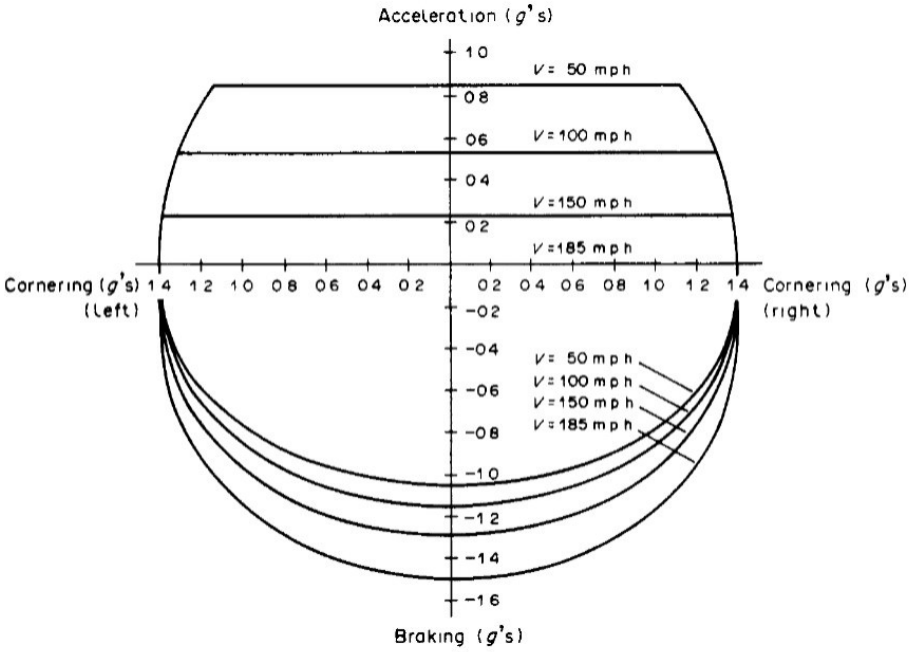

Figure 2.1: g-g diagram of F1 racing car [4].

- **Dynamic vehicle model with free trajectory**: These methods started to be common published around the 2000s, since the evolution of optimal control problems solving was strongly linked to the computational capacity. Moreover specific algorithm have been developed during these years and they have been progressively applied to MLT problems [3].

  Instead of using a simple point-mass car, a more sophisticated approach involves adopting a complex vehicle model even with non-linearity (see Section 2.2.3 for a more detailed description of the adopted vehicle models.). This provides more reliable representations of real-world vehicle behaviour. However, the increased complexity also presents challenges, thus the mathematical methods adopted to solve such complex equations are not easy to handle and require an adequate knowledge of the theory behind them. Among the procedures adopted, three major groups can be identified and they are explained into Section 2.2.2.

  Each mathematical method necessitates its own unique procedure, making it difficult to find a common approach that addresses all methods compared

to the QSS with fixed trajectory methodology proposed in Subsection 2.2.1. Nevertheless, any combination of these approaches published so far can be restructured by integrating the contents from the specific vehicle model section (Section 2.2.3) and the mathematical methodology described in Section 2.2.1.

- **QSS with free trajectory**: The concepts of using g-g diagrams (numerically or experimentally computed) as tools for modelling the vehicle handling capability well suits the QSS approach, since it permits to define the vehicle maximum acceleration (both longitudinal and lateral) for every steady-state evaluated. This concepts does not prove it wrong even in the approach discussed in this subsection, since it can be applied no matter the nature of the trajectory, if it has been calculated or deduced. Moreover, the g-g diagrams have the advantage to summarise all the complex, non linear vehicle behaviours, lightening the computational effort over the OCP solver. For this reason adopting a QSS with free trajectory seems to be the most promising solution, since it allows to handle a reliable vehicle model with a limited increase of complexity and computational requirements [10]. Combining the advantages of a QSS approach with the adaptability of not having any reference trajectory one seems to create the right compromise among the possibilities presented for the MLTS OCP.

  Respect to the other two approaches, a QSS with free trajectory is 10 times faster than the free-trajectory approach using a dynamic model, and about 10 times slower than the predefined trajectory QSS approach [10].

  Considering its advantage to be light but reliable solution, a QSS with free trajectory approach has been adopted during the project of this thesis. A lap time simulator has been chosen as a plant inside a control system whose goal was to run simulations until a satisfactory trajectory has been deduced. To deepen the vehicle model used, so as the algorithm used to simulate a lap time, reference can be made to section 4.

2.2.1 Predefined (fixed) and free trajectory

The difference in using a fixed trajectory approach over a free-one is due to the goal set by the MLT problem. In fact, a fixed one is mostly related to simulate as

close as possible a lap time, the other approach tries to apply an OCP to a lap time, meaning that it has to compute an optimised trajectory without having a reference. Therefore there is a notable difference among the two approaches and for this reason the selection of one of the two is fundamental and strikingly related to the requirements of the problem set.

- **Fixed trajectory**: In early stages the concept of lap time simulation (MLTS) was not to directly calculate an optimised trajectory but to replicate as realistically as possible a real lap time. This was done by reverse engineering, where the driver inputs needed to replicate a determined trajectory were calculated. In particular most applications replies the commands of a professional diver, being accelerations commands and steering wheel actions [4]. To solve such problem the idea is to maximise the performance of the vehicle along that trajectory, which physically consist on creating the best possible velocity profile [3].

  The apex-finding method is the method most adopted and consist on the identification of the corner apex along the predefined trajectory, also nominated as critical point (make reference to figure 2.2).

  At this point the vehicle is supposed to be at its maximum acceleration capacity, meaning that its velocity can be computed form a lateral equilibrium equation, since the radius of curvature is known by the trajectory information. This idea can be simply formulated as follow:

$$m\frac{{v^2}_{\max}}{\rho} = \mu m g \Rightarrow v_{\max} = \sqrt{\mu \rho g} \tag{2.2}$$

  Having a defined speed at the apex, the car at the entrance of the curve is supposed to be hitting its maximum braking capacity, meaning that is on the negative apex of the g-g diagram envelop. Having a defined maximum braking acceleration, the velocity profile of the car can be easily deduced by integration. Therefore the velocity profile is calculated step by step during the deceleration phase until or a physical constrain is exceeded or the deceleration curve intersect an acceleration curve. In fact the same concept, applied forward, can be adapted to the curve exit phase. With the hypothesis that at the exit of the curve the car is accelerating at its maximum, the speed profile is computed

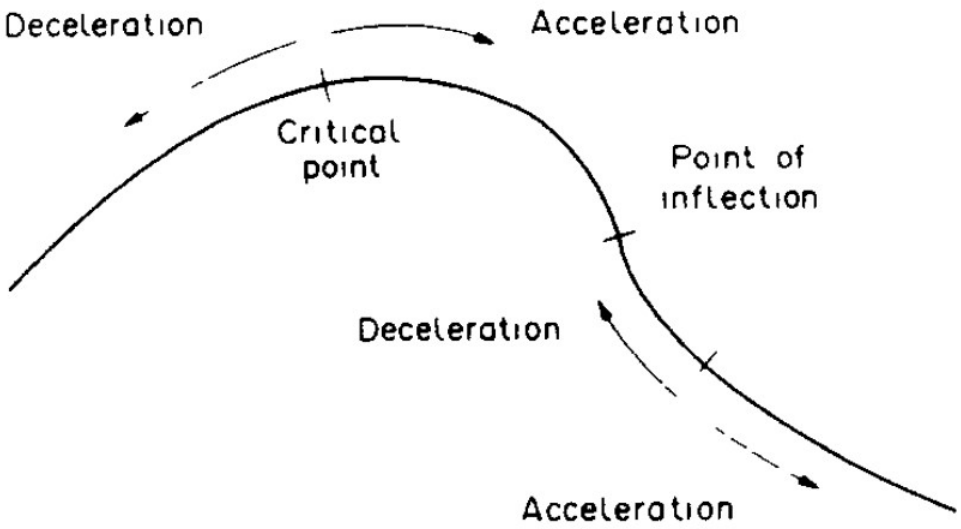

Figure 2.2: Apex-finding technique [4].

by integration. The velocity profile computed in two consecutive apexes are then connected among them at their intersection point. The physical limits of the vehicle can be modelled by g-g diagrams, avoiding to implement complex vehicle models into the simulation [4].

- **Free trajectory**: As the name suggest, this procedure involves the creation of trajectories from scratch. For this reason it involves the implementation of optimisation process, where a dynamic parameter as the trajectory or the curvature is set as the minimisation argument. The first application of such method adopts a procedure which is not so different from the methodology interpreted by the fixing trajectory, where peculiar points of the trajectory are identified and the trajectory is evaluated form these [11]. In particular, four points are set when cornering:
  - Turning point: where the driver starts to act on the steering wheel;
  - Brake releasing point: where the car does not generate anymore longitudinal acceleration;
  - Apex: car is in pure lateral acceleration condition after a coasting period;

– Corner exit: point where the steering angle imposed by the driver returns to a null value.

By the definition of these four points, the trajectory can be computed by an interpolation involving a cubic spline. The optimisation is carried out by a genetic algorithm whose purpose is to find the combination of the four points such that minimises the cornering radius. This process is well represented in the following figure 2.3.

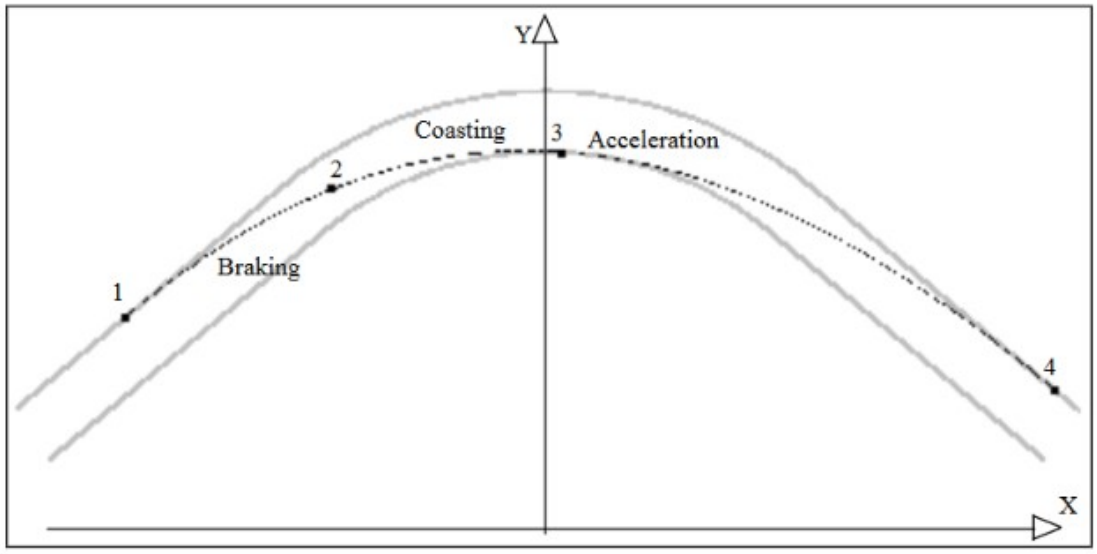

Figure 2.3: Points identification for free-trajectory approach [11].

2.2.2 Optimisation problem

During the years several approaches have been considered regarding the optimisation problem, one of which regards the argument of the optimisation problem. Most of the literature propose the most intuitive approaches being the lap time and the curvature, which both needs to be minimised. Intuitively the two approaches should provide comparable results, however only in the latter years a direct comparison of the two approaches have been studied [12]. The conclusion of such analysis is that configuring the same optimisation problem where the only difference is the argument choose, lap times and the trajectory itself changes accordingly. Moreover

the amount of changes is quite significant, being it of 1.23 *seconds* especially considering that the track analysed (Airport of Berlin E-prix of 2018) is quite short (776 *meters* in total). The results are shown in the figure 2.4 where it is possible to appreciate the difference not only in the trajectory but also in the lap times reported in the legend.

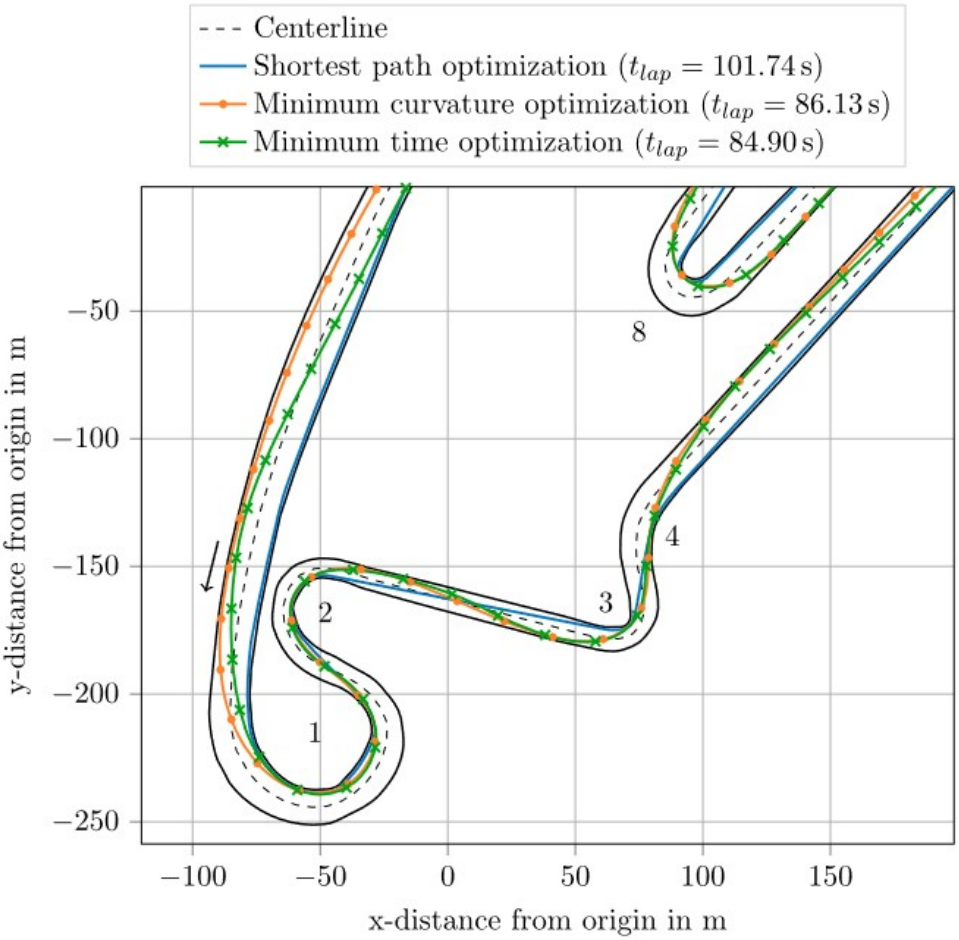

Figure 2.4: Solutions for different optimisation arguments [12].

Also Game changing is the methodology adopted for solving the optimal control, in other words how the optimisation is computed and solved. The main idea is to transform the minimum lap time problem into an OCP, where the equations of motion act as constraints in the optimisation process. There are three primary

categories of methods for solving OCPs: dynamic programming (DP), indirect optimal control, and direct optimal control [1].

Dynamic programming has theoretical advantages, such as handling discrete/-continuous variables and guaranteeing the global optimum, but it suffers from the curse of dimensionality, especially with larger vehicle models. To address this limitation, differential dynamic programming (DDP) was introduced, which solves a sequence of quadratic subproblems obtained from the quadratic approximations of the objective function around a reference trajectory. Hybrid DDP is an advanced version that combines DDP with nonlinear mathematical programming techniques, showing promise in solving complex trajectory optimization problems.

Indirect methods rely on the Pontryagin Maximum Principle to determine the necessary conditions optimally. They involve a set of ordinary differential equations with initial and final boundary conditions, forming a two-points boundary value problem (TPBVP) coupled with a minimisation problem to derive the optimal control law. Various numerical techniques are used to solve such problems, and several works have applied this approach to minimum lap time simulations for both motorbike and car scenarios.

Direct methods translate the OCP into a discrete constrained minimisation problem, known as nonlinear programming problem (NLP). There are different ways to discretise the controls and states, such as sequential discretisation or collocation. Among the sequential discretisation methods, the direct multiple shooting method emerged as the most efficient, as it is less affected by high sensitivities and is naturally suited for parallelization. This method has been applied successfully to minimum time problems, including gear choice, using software like MUSCOD-II and partial outer convexification to handle discrete variables.

2.2.3 Vehicle Models

The state of the art uses different vehicle models according to need. In fact, it is possible to find models from the simplest to facilitate computation, to more

realistic models where much more importance is given to the representative capacity of the model [4]. In addition, a road often taken is to use simple models but incorporating in these features describing more complete dynamics (e.g: tyre temperature, cornering stiffness variation, etc... [3]).
In this thesis two different vehicle model have been adopted, one using a point-mass representation, the other is a complete 8-degree-of-freedom model, which provides an accurate description of all the dynamics of the vehicle in motion. In the first phase of the project, where the objective was to implement an RL capable of generating and testing a trajectory on a lap time simulator such as OPENLAP, the latter adopts a point mass model using g-g maps to establish the vehicle manoeuvrability limit. Subsequently, having the objective of implementing an architecture that allows the RL to learn through data accumulated from previous manoeuvres, the model to be considered was necessarily changed. In fact, it is fundamental that the information coming from the vehicle are as reliable as possible, not only for a question of the quality of the agent's action, but also to facilitate the training of the NN. The agent, in this way, having more reliable data will be more inclined to generate control actions (or path re-planning as in the evaluated case) of higher quality, thus facilitating the implementation of the AI in the architecture [13]. Hence the need for a more complete high fidelity vehicle model than a point mass model.

Point Mass Model

The point mass model is a straightforward representation based on Newton's laws, featuring three degrees of freedom: longitudinal velocity, lateral velocity, and yaw motion. Figure 2.5 illustrates the body diagram of this model.

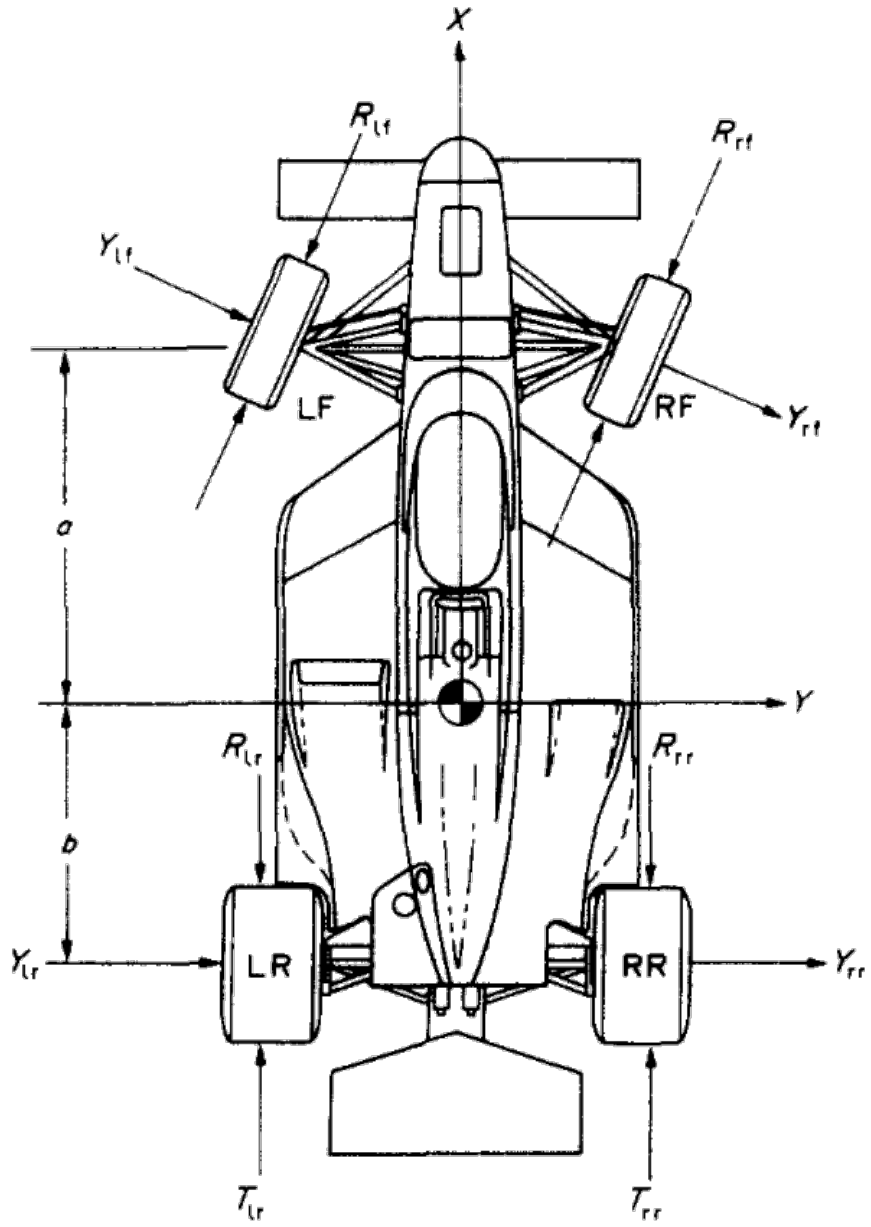

Figure 2.5: Point mass vehicle model simplification [4].

The model is applicable to open-wheeled racing vehicles with ground effects aerodynamics, rear-wheel drive, and rear-engined configurations. It assumes rigid suspension systems, minimising suspension effects without significantly compromising accuracy. The equations of motion for this model are as follows:

$$u = \frac{1}{m}\left[T_a + T_b - f(mg + N_a) - R_a - \delta C_r(\delta - \frac{v + ar}{u})\right] + vr$$
$$v = \frac{1}{m}\left[Cf(\delta - \frac{v + ar}{u}) + C_r(\frac{v - br}{u})\right] - ur$$
$$r = \frac{1}{I_{zz}}\left[-fmhur + aC_r(\delta - \frac{v + ar}{u}) - bC_r(\frac{v - br}{u})\right]$$

Where: - u is the longitudinal velocity along the X axis, - v is the lateral velocity along the Y axis and - r is the yaw rate.
The point mass model offers rapid computation and robustness, making it suitable for various applications, including those involving complex multi-body models. However, it neglects transient behaviours like tire load dynamics, yaw dynamics, and suspension damper effects, making it less accurate compared to more sophisticated techniques [1].

8 Degree-of-Freedom (8 DOF) Model

The 8 DOF model is one of the most complex and accurate vehicle models available. It accounts for longitudinal, lateral, yaw, and roll motions, as well as the dynamics of all four wheels. It models forces (lateral and longitudinal) and rolling resistance using Pacejka Magic Formula.

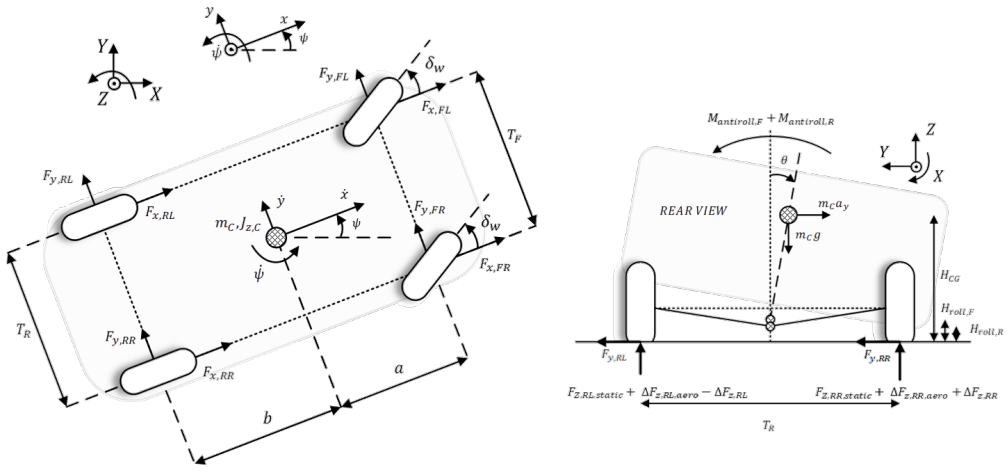

Figure 2.6: 8 DOF vehicle model [14].

The equations of motion for the 8 DOF model are as follows:

$$\ddot{x} = \frac{1}{m_C}\left(F_{x,FL} + F_{x,FR} + F_{x,RL} + F_{x,RR}\right) + \dot{y}\dot{\psi}$$

$$\ddot{y} = \frac{1}{m_C}\left(F_{y,FL} + F_{y,FR} + F_{y,RL} + F_{y,RR}\right) - \dot{x}\dot{\psi}$$

$$\ddot{\psi} = \frac{1}{J_{z,C}}\Big(a(F_{y,FL} + F_{y,FR}) - b(F_{y,RL} + F_{y,RR}) + \frac{T_F}{2}(F_{x,FR} - F_{x,FL}) + \frac{T_R}{2}(F_{x,RR} - F_{x,RL})\Big)$$

$$\ddot{\theta} = \frac{1}{J_{x,C}}\left(m_C a_y (H_{CG} - H_{roll,F})\cos(\theta) + m_C g (H_{CG} - H_{roll,F})\sin(\theta) - (F_{y,RL} + F_{y,RR})(H_{roll,R} - H_{roll,F}) - M_{antiroll,F} - M_{antiroll,R})\right)$$

$$\dot{\omega}_{FL} = \frac{1}{J_{y,w,F}}\left(T_{FL} - F_{l,FL}R_{w,F} - M_{y,FL}\right)$$

$$\dot{\omega}_{FR} = \frac{1}{J_{y,w,F}}\left(T_{FR} - F_{l,FR}R_{w,F} - M_{y,FR}\right)$$

$$\dot{\omega}_{RL} = \frac{1}{J_{y,w,R}}\left(T_{RL} - F_{l,RL}R_{w,R} - M_{y,RL}\right)$$

$$\dot{\omega}_{RR} = \frac{1}{J_{y,w,R}}\left(T_{RR} - F_{l,RR}R_{w,R} - M_{y,RR}\right)$$

$$\dot{X} = \dot{x}\cos(\psi) - \dot{y}\sin(\psi)$$

$$\dot{Y} = \dot{x}\sin(\psi) + \dot{y}\cos(\psi)$$

$$\dot{s} = \frac{1}{1 + k_{\text{ref}}e_y}\left(\dot{x}\cos(e_\psi) + \dot{y}\sin(e_\psi)\right)$$

2.3 MLTS procedure: a general methodology

As discussed in the previous section (2.2), various methodologies have been developed over the years for simulating and optimizing lap times. The process of simulating and minimizing lap times is inherently an optimization problem that doesn't adhere to a strict or predetermined sequence. However, some common-sense principles can be identified. In particular, the methodology here presented offers a valuable approach that can be applied to a wide range of applications [3].

1. **Track Modelling and Construction**:

The initial and crucial step in setting up a simulation is the creation of an accurate track model. Without a faithful representation of the environment, meaningful simulations are unattainable. The manner in which the circuit is depicted plays a pivotal role in ensuring a realistic simulation. Essential parameters such as circuit shape, track curvature, and track-width details are indispensable for establishing inequality constraints that keep the vehicle on the track. It's noteworthy that, as far as the author is aware, there exists no standardised or widely accepted database for race track representations. Therefore, all the circuit used in this work are inherited from previous works and online sources as Github.

2. **Vehicle Modelling**:

   Irrespective of the chosen approach or application for lap time calculation, it's imperative that the vehicle model accurately represents the capabilities of the car. Vehicle modeling entails the establishment of equations of motion and kinematic differential equations that delineate the relationship between system states and controls. This model forms the foundation for the optimal control algorithm, allowing the vehicle to navigate the track and determine an optimal control strategy.

3. **Cost Function Definition**:

   In cases where the Minimum Lap-Time Simulation (MLTS) includes trajectory optimisation, defining an appropriate cost function becomes a pivotal step. The cost function, minimised during the optimisation process, must be meticulously defined to align with the simulation objectives. Additionally, specifying the constraints applied to the cost function holds equal importance. A standard mathematical representation of this problem is detailed in Equation (2.1).

4. **Constraints on Vehicle Dynamics**:

   The optimisation problem typically seeks to minimise lap times while adhering to constraints imposed by the vehicle's dynamics, initial conditions, track limitations, and consistency of both states and inputs. These constraints may encompass limits on steering torque, steering angle rate of change, suspension travel, and engine power.

Minimum Lap Time Optimisation is a mathematical optimisation problem where the mathematical constraints imposed are the vehicle's equations of motion. In this way, whatever trajectory is calculated, there is a certainty that the vehicle can actually perform the given trajectory. Hence the importance of imposing constraints that are as realistic as possible, in other words, providing equations of motion of the vehicle that are as complete as possible. Since these equations are impositions on an optimisation problem, the level of detail of the equations of motion has a physiological limit, in the sense that one could even impose a model with high DOF, but with the certainty of incurring either extremely high computation times or a mathematical divergence, i.e. the optimisation algorithm cannot find a solution to the problem posed. Hence the tendency in the literature to find a compromise between simplified models and the addition of particular dynamic features, as described in the section 2.2.3. Generally, vehicle dynamics constraints on steering torque, steering angle rate of change, suspension travel, and engine power [1].

5. **Solving the Optimisation Problem**:

   Following the formulation of the optimisation problem with its defined cost function and constraints, the next step is to solve it. Various methods have been proposed, and the current state-of-the-art often relies on Non-Linear Programming techniques. Notable examples include the MATLAB toolbox ICLOCS [15], Pontryagin's indirect method using Maple software [16], and IPOPT [17].

2.4 State of art: a summary table

During the course of the project, a careful analysis of the state of the art was made in order to assess the most commonly used technologies in the field of MLT optimisation, while also trying to catch a glimpse of possible new solutions not yet fully explored by current research. Several papers were read and a summary table was drawn up (2.1 up to 2.6), in order to facilitate consultation and recall of the main papers on the subject of minimum lap time optimisation, with emphasis on the methodologies used, how the MLT problem was modelled and which vehicle

model was used.

In short, it can be seen that in the early stages, the main objective was to try to replicate a lap time as faithfully as possible, while only in recent years has an attempt been made to exploit NLP to directly calculate new optimised trajectories, perhaps even using lap time simulators to optimise the design of certain vehicle components.

Artificial Intelligence approaches, on the other hand, are of the latest generation, where mainly conventional deep learning networks are used to model the vehicle model, while Reinforcement Learning is more involved by exploiting its model-free capability. In particular, it is worth mentioning how RL seems to fit best in full end-to-end driving contexts, i.e. the mere presence of a Deep Neural Network between the sensors and the control action (also referred to as the black-box approach). In this way, the process of feature extraction and control is all in the hands of the AI, which extracts features directly from the camera images, these are utilised by the AI (mainly RL) which directly generates control actions, without receiving any information about the vehicle dynamics. This is enabled by the RL's ability to learn from interaction with the environment alone, based on which it receives a reward or penalty. Since the RL agent learns based only on maximising the reward it receives, it can learn policy without having any information about the model, based only on its interaction with the environment.
Below are some tables summarising the state-of-the-art study carried out.

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
Gadola et al.	1996	First Lap Time Simulator developed. Optimal inputs calculated in order to find the best lap time	Point Mass model with Pacejka 1989 tyre model and lateral load transfer distribution considered	Optimisation process	Genetic Algorithm
Hendrikx et al.	1996	Gradient method applied to MLTS with Optimal Control Theory	Point mass model with Pacejka 1989 tyre model formulation	Optimisation process	Gradient method
Cossalter et al.	1999	MLT approach as an Optimal Control Problem for motorbikes	Motorcycle model with 4 DOF	Optimisation process	Gradient method
Casanova et al.	2000	MLT with Optimal Control strategy	Point Mass Model with 7 DOF (roll axis position, the roll stiffness distribution and the mass centre height)	Optimisation process	Parallel shooting method
Brayshaw et al.	2005	QSS approach to simulate set up changes	7 DOF vehicle model	QSS optimisation	*fmincon* in MATLAB
A. Rucco et al.	2012	Division of longitudinal and lateral dynamics, the MLT optimisation is handled as a fixed horizon constrained problem	single-track rigid car which includes tire models and load transfer (LT-CAR model).	Optimisation process	PRONTO approach
Julian P. Timings et al.	2012	New approach to formulate the MLT optimisation: linearizing the vehicle's trajectory relative to the track meaning computationally efficient convex quadratic programming problem	Linear Time variant Bicycle model with Pacejka 1989 formulation	Optimisation process	Convex QP problem

Table 2.1: References: [11], [18], [19], [20], [21], [22], [23], [24].

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
Paul A. Theodosis et al.	2012	Trajectory found by optimisation process where straights, clothoids and circular arcs can be used emulate the racing line professional race car given as starting point	Vehicle model not implemented: handling limit estimated through g-g diagrams	Optimisation process	Sequential gradient based approach
G. Perantoni et al.	2014	Minimum-lap-time optimal control of a Formula One car with simultaneous parameter optimisation	7 DOF vehicle model	Optimisation process	Direct collocation scheme and a NLP algorithm (MATLAB-ICLOCS)
R. Lot et al.	2015	MLT problem solved through optimisation process and compared with on-track data	7DOF go kart model	Optimisation process	Maple for writing the equations of motions; Indiretch method ofr NLP (Xoptima and Mechatronix)
Brunner et al.	2017	MPC utilizes information from previous laps to enhance the performance with continuous operation on the race track. Tested on 1:10 cars experiments.	Simple bicycle model (3 DOF) for lighten MPC	Repetitive Learning MPC	Python + Julia code using ROS. Optimisation problem solved through IPOPT
D. Caporale et al.	2018	Fully AI driver, where the path planning is done with the objective of reducing the lap time. ONLINE TEST ON HARDWARE.	MPC based on a simple 3 DOF vehicle model	MPC whose optimisation problem is solved through MATLAB Optimization Toolbox	MLT is performed as a convex combination of minimum path optimisation and maximising the speed profile.
J. Kabzan et al.	2019	Online learning data-driven Model Predictive Controller. ONLINE TEST ON HARDWARE.	Nominal vehicle model is a bicycle model with nonlinear tire forces (3 DOF)	Learning MPC	MPC relies on simple nominal vehicle model which is improved based on measurement data and tools from machine learning.
A. T˘atulea-Codrean et al.	2020	Design of a Non Linear MPC (NMPC). Optimal control strategy given to a NN which is used to replace the MPC for online implementation. ONLINE TEST ON HARDWARE.	Bicycle model (3 DOF)	NN replacing NL-MPC	Farthest visible point in the environment as the goal point, the NMPC calculates the shortest distance among the actual point and the goal point.

Table 2.2: References: [15], [16], [25], [26], [27], [28], [7].

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
Massaro et al.	2020	OCP where the vehicle model is a QSS for fast computation	3DOF model constrained to move within the road borders satisfying the limits of the g-g-speed surface	OPC	For a given g-g diagram, OCP run with IPOPT considering the QSS model. Nonlinearities of the model considered within the g-g surface limits
L. Leonelli et al.	2020	MLT applied to motocycles. Comparison of the result with experimental results .	Motorcycle 6DOF model	OPC	MLT handled as OCP using GPOPS-II MATLAB software. Peculiar motorbikes dynamics considered (wheelie and stoppie, restriction on tyre normal load, zero torque to frontal tyre, roll angle limitations and quadratic terms in the cost function for manouvrability.)
A. Heilmeier et al.	2020	Software stack capable to compute the optimal trajectory of a given circuit. MLT considered as a Minimum Curvature Trajectory optimisation problem.	Vehicle limits modelled with g-g diagrams. Point mass model used for the velocity profile generation.	OCP	The minimum curvature path is generated using a quadratic optimisation problem (QP) formulation with iterations. Velocity profile calculated considering the limits of longitudinal and lateral accelerations.
A. Jain et al.	2020	New mathematical approach for optimising the lap time using Bayesian optimization	Feasibility of the trajectory checked considering a friction ellipse (g-g diagram)	OCP	Giving centreline waypoints (x,y), track width and three vehicle characteristics (data-driven approach), this new algorithm calculate the optimal trajectory using BayesOpt
L. Leonelli et al.	2020	MLT applied to motocycles. Comparison of the result with experimental results .	Motorcycle 6DOF model	OPC	MLT handled as OCP using GPOPS-II MATLAB software. Peculiar motorbikes dynamics considered (wheelie and stoppie, restriction on tyre normal load, zero torque to frontal tyre, roll angle limitations and quadratic terms in the cost function for manouvrability.)

Table 2.3: References: [3], [17], [12], [29], [6].

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
U. Rosolia et al.	2020	Non-linear Learning MPC (LMPC) which learns from previous laps subset of stored data.	Process of linearisation of the model: The estimated nonlinear dynamics are linearized over time-varying regions of the state space to construct an Affine Time-Varying (ATV) model	OCP	LMPC formulated as a Quadratic Program thanks to linear modellisation around the current position using local linear regressor. At each lap data is collected and feed to the next lap to the LMPC, which learns to optimise the lap time by iterative trials.
Capo et al.	2020	Develop some agents capable to compute a target point for the vehicle which is then turned into actionable commands by a custom low-level logic.	Default TORCS vehicle model (7 DOF vehicle model programmed in C++)	Deep Reinforcement Learning	Reward function includes variation of the distance raced, negative rewards to discourage going out of track, colliding with walls and driving backwards
S. Garlick et al.	2021	Train a NN to replicate as smooth as possible the optimal trajectory from other optimisation process	Point Mass Model (coming from the optimisation process that they referred as input to the Neural Network)	NN	The NN is tested only as matter of rms distance from the optimizer results, there is no actual test over a laptime
F.Christ et al.	2021	MLT which takes into account wheel-specific tyre-road friction coefficients along the racetrack using a track friction map	single and doubletrack model with QSS tyre load simplification and non linear tyre model	OPC	MLT considered as OCP converted into NLP using direct orthogonal Gauss-Legendre collocation and solved through IPOPT
E.Chisari et al.	2021	Pure simulation training adopted into small scale vehicle application. SAC (Soft-Actor-Critic) RL adopted.	Bicycle model with Pacejka tire models, but they use model randomisation to train an agent capable to adapt to real scenarios	Deep Reinforcement Learning	Dense reward function used to penalise constrains violations. RL agents how to steer to obtain the lowest lap time as possible. However is over-performed by state-of-art MPC
F.Fuchs et al.	2021	Soft-Actor-Critic RL which learns to drive at the limits from image processing.	Model-free: The RL learns directly acting on the steering wheel thanks to the information extracted from video processing	Deep Reinforcement Learning	The MLT is formulated with the reward function, which tries to maximise the distance travelled in consecutive fixed time defined sectors.

Table 2.4: References: [30], [31], [32], [33], [6].

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
P. Cai et al.	2021	End-to-end autonomous racing driving. Combination of Imitation Learning and RL with DDPG algorithm. Two-step learning: IL used as pre--training, further exploration done by DDPG	Model free: End-to-end driving means that the RL gets information from camera processing and directly acts on the inputs	Imitative and Deep Reinforcement Learning	Imitation Learning over human inputs. DDPG encouraged to find better lap times exploiting a reward function with three costs: speed cost, collision cost, uncertainty cost (due to image processing)
O. Borsboom et al.	2021	Analysis over transmission configurations to achieve the best configuration for lap time minimisation of electric racing cars. TESTED ON 1:10 CARS	Longitudinal Dynamics only since the focus is transmission design	OCP	Minimum-lap-time control problem in a convex fashion for a given driving path and compute its globally optimal solution with second-order conic programming algorithms.
J. Klapalek et al.	2021	Reformulate the MLT control problem as an unconstrained problem via coordinate transformation by using proposed homeomorphic mapping, also called Matryoshka mapping algorithm. TESTED ON 1:10 CARS	Single-track model for speed profile	OPC with Genetic Algorithm	Initially the racing line is estimated via backward direction, then facing forward the velocity profile is selected to comply with maximum lateral acceleration, evaluated considering cars parameters.
A. Brunnbauer et al.	2021	Model-Based RL which learns an approximated dynamics model in an unsupervised fashion by only having access to LiDAR observations.	Instead of using traditional state-space representation of vehicle models, an abstract state representation and a transitional model are deduced instead.	Deep Reinforcement Learning	DREAMER (problem formularised as Partially-Observable Markov Decision Process - POMDP). MLT performance through reward function that maximises the distance travelled.
A.Remonda, et al.	2022	Analytical study over different RL configurations to establish the best RL algorithm for autonomous racing.	Model-free RL: inputs are telemetry data and the output are control actions over steering and acceleration.	Deep Reinforcement Learning	MLT formulated as maximising the velocity profile.
S. Broere et al.	2022	Minimum lap time optimisation used for powertrain design	Two-track model	OCP	MLT formulated as optimal control problem in a convex fashion solved with off-the shelf second-order conic programming algorithms delivering a globally optimal solution.

Table 2.5: References: [34], [35], [36], [37], [38],[39].

Name and Reference	Year	Description	Vehicle model	Category	How the MLT is handled
P. de Buck et al.	2022	Minimum lap time modelled as a QSS fixed-trajectory problem for exploration of optimal torque vectoring controls and vehicle's active aerodynamic performance evaluation	3-DOF car model	OPC	MLT OCP turned into NLP problem, solved with IPOPT algorithm
X. Hou et al.	2022	Residual RL for planning and control. Expert driver manoeuvres given as benchmark from which the agent learns how to further improve	Double track for modelling a base policy. No model used for trajectory planning, just evaluating maximum handling performance	Deep Reinforcement Learning (SAC algorithm)	Expert driver skills improved thanks to exploration of the agent in control considering g-g diagrams as limit function for their operations.
K. Tucker et al.	2022	MLT optimisation formulated as optimal control problem with a new QSS vehicle model	7 DOF vehicle model with magic formula tyre modelling	OPC	MLT modelled with an optimisation problem, maximising the steady-state acceleration magnitude which is computed using NTA surface function.
M. Piccinini et al.	2022	Online minimum-time motion planning, where a high level NMPC compute trajectories and low level NN and long. Controller produces command actions for following the calculated trajectory	14 DoF model (trajectory planning)	NMPC for OCP and NN (LSTM) for control action (from states a delta is computed)	OCP modelled with tunable soft initial and final conditions for states in addition to the minimisation of the lap time
J. Biniewiczand et al.	2023	QSS model for motorcycle MLT optimisation through the use of semi-empirical g-g diagrams	Rigid-body motorcycle model with 3DOF	OPC	MLT modelled as OCP and solved through NLP with IPOPT algorithm

Table 2.6: References: [40], [5], [41], [42], [43], [22], [23].

Chapter 3

Reinforcement Learning

3.1 Introduction

Deep reinforcement learning (DRL) is a type of machine learning that allows computers to learn how to behave in an environment by trial and error. DRL algorithms use deep neural networks to learn from their experiences and improve their performance over time.
DRL algorithms are different from other machine learning algorithms in a few ways:

- They are trained on sequential data, which means that they learn from a sequence of actions and rewards.
- They are trained in an environment, which means that they learn by interacting with the world around them.
- They are trained using reinforcement learning, which means that they learn by trial and error.

DRL algorithms have been used to achieve state-of-the-art results in a variety of tasks, including playing video games, controlling robots, and trading stocks.
Here is a simpler analogy:
Imagine a child learning to walk. The child starts by crawling around and trying different things. When they take a step and fall down, they receive a negative reward. When they take a step and stay upright, they receive a positive reward.

Over time, the child learns to walk by trial and error, and they maximise their total reward by staying upright.
DRL algorithms work in a similar way. They start by taking random actions in an environment and observing the rewards they receive. Over time, they learn to take actions that lead to higher rewards.
DRL is a powerful tool that can be used to create intelligent machines that can learn from their experiences and improve their performance over time [44].

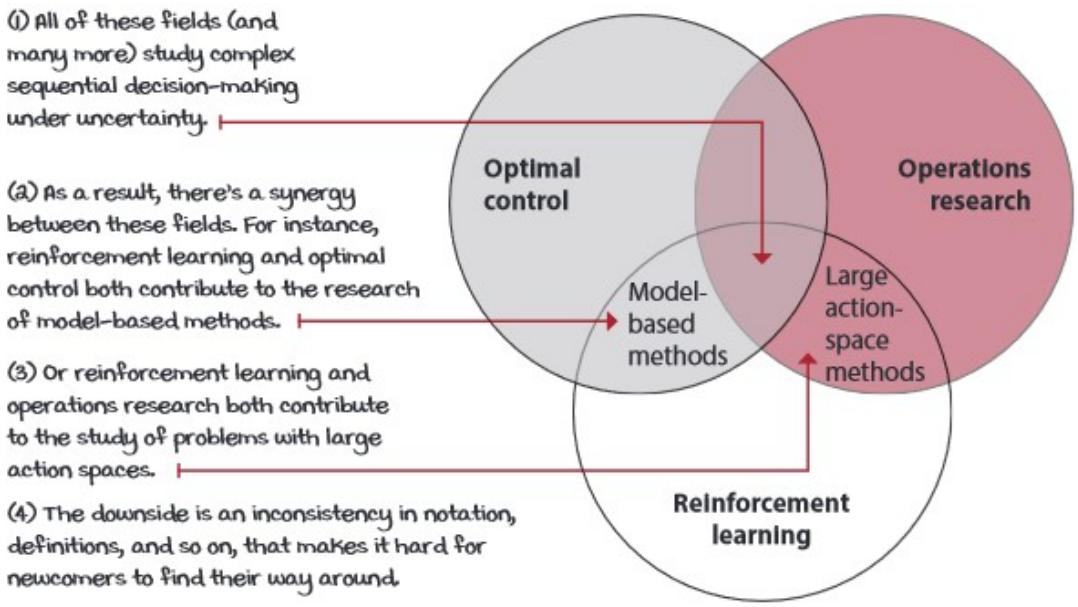

Figure 3.1: RL and its relation with control filed [13].

Remark: Deep NN definition
It should be pointed out that the concept of Deep when referring to a Neural Network is somewhat ambiguous. In fact, there is no canonical and accepted definition of the limit beyond which the neural network is sufficiently long to be considered deep. There is no definitive answer to the question of how many layers a neural network needs to have in order to be considered deep. However, it is generally accepted that a network with two or more hidden layers can be considered deep. A network with only one hidden layer is typically called "shallow."
Some people believe that the term "deep" is relative, and that the number of layers needed for a network to be considered deep will increase over time as our

understanding of neural networks improves. For example, in ten years, people might think that anything with less than ten layers is shallow.
Informally, the term "deep" is used to describe neural networks that are difficult to train and understand. This is because deep networks have many parameters, and it can be difficult to determine how these parameters interact with each other to produce the desired output.
Here is a simpler analogy:
Imagine a neural network as a series of rooms, with each room containing a group of neurons. The input layer is the first room, and the output layer is the last room. The hidden layers are the rooms in between.
Shallow neural networks have only a few rooms, while deep neural networks have many rooms. The more rooms a neural network has, the more complex the functions it can learn. However, the more rooms a neural network has, the more difficult it is to train and understand [45].

3.2 Definitions

The main operation of the RL is that given a given state in which the agent finds itself, it will attempt to perform the action such that the long term reward is as high as possible.
A reward is a number that tells an agent how good a transition is. In reinforcement learning, an agent observes a state, takes an action, receives a reward, and transitions to a new state. The reward is a single number that indicates how good the transition was for the agent. An adequate definition of the reward function is therefore fundamental, since it is the criterion upon which the agent learns the control problem posed. Its mathematical definition can be formulated as:

$$r(s, a) = \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right] \tag{3.1}$$

Meaning that the reward r is a function that for a state-action pair (s, a), it is the expectation $\mathbb{E}$ of the reward of a given time step R_t, given the action and the state of the previous time step S_{t-1}, A_{t-1}.
However, making reference to the control problem, what the agent learns by the reward is a policy, which means a series of actions aimed at maximising future

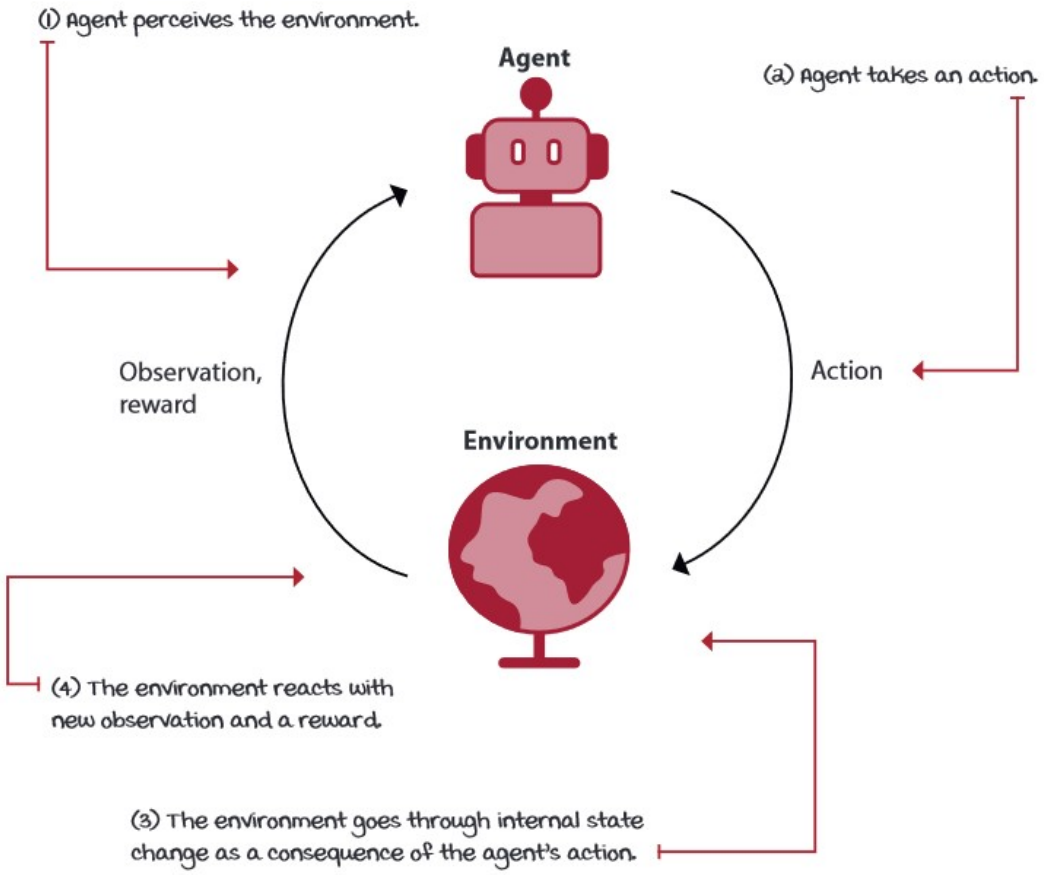

Figure 3.2: RL and its interaction with environment [13].

reward and involving optimal control behaviour.

The action-value function, also known as the Q-function, is a function that estimates the expected return of taking a particular action in a particular state (figure 3.2 schematise this idea). It is a key concept in reinforcement learning, which is a type of machine learning that allows computers to learn how to behave in an environment by trial and error.

The Q-function can be used to improve policies, which are rules that tell a computer what action to take in a given state. By comparing the Q-values of different actions in the same state, it is possible to select the action that is expected to lead to the highest reward.

The Q-function also captures some of the dynamics of the environment, which

means that it can be used to improve policies without the need for a model of the environment. This is an important advantage of the Q-function over other methods for reinforcement learning [13]. Its mathematical description can be formulated as following:

$$q(s,a) = \mathbb{E}_{\pi}\left[R_t + \gamma G_{t+1} | S_{t-1} = s, A_{t-1} = a\right] \tag{3.2}$$

The Q-value of an action a is the average reward that an agent can expect $\mathbb{E}$ to receive if it takes that action a in a particular state s and then continues to follow its policy π.
The practical way in which a given policy is undertaken by the agent is based on reinforcement techniques [46]. Indeed given a given policy, defined as:

$$\pi_{\theta} = P\left[a|s,\theta\right] \tag{3.3}$$

Where θ are the policy parameters, which, during training, the neural network will adjust these parameters so that the agent's behaviour is considered satisfactory.
In order to evaluate the goodness of the policy selected by the agent, Policy Objective Function are defined, whose purpose is to measure how well a policy is doing at achieving this goal. An intuitive policy objective function can be represented by:

$$L(\theta) = \mathbb{E}_{x\sim p(x|\theta)}\left[R\right] \tag{3.4}$$

Where the expectation $\mathbb{E}$ of the total reward R is calculated under some probability distribution $p(x|\theta)$ are parameterised by some θ.
Considering the Q-function, which estimates the return of a certain set of actions, $Q(s_t, a_t) = R_{t+1}$ the gradient of the a deterministic policy can be stated as follows:

$$\frac{\partial L(\theta)}{\partial \theta} = E_{x\sim p(x|\theta)}[\frac{\partial Q}{\partial \theta}] \tag{3.5}$$

Actor Critic algorithm

The RL studied in this thesis exploit an Actor-Critic algorithm which is a hybrid method that combines the policy gradient method and the value function method. The policy function is called the actor, and the value function is called the critic. The actor produces an action given the current state of the environment, while the critic produces a signal to criticise the actor's actions. The actor network learns to

take actions that maximise the expected cumulative long-term reward, as predicted by the critic network. The critic network learns to accurately predict the expected cumulative long-term reward for each action-state pair.
An analogy with an everyday context can be helpful. An actor is like a junior employee who does the actual work, while a critic is like a boss who criticises the work and hopefully helps the junior employee improve.
In this way, by substituting the Q-function with a NN, meaning that $Q(s, a, w)$, where w are the weight of the Neural Network, the **Deep Deterministic Policy Gradient** formula can be derived:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\partial Q(s, a, w)}{\partial a} \frac{\partial a}{\partial \theta} \tag{3.6}$$

Deep Deterministic Policy Gradient (DDPG)

The algorithm used in this thesis is the Deep Deterministic Policy Gradient (DDPG), a model-free online and off-policy reinforcement learning method. Each characteristic has a particular meaning:

- Model-Free: It learns without knowing the model of the environment.
- Online: Its learning process is continuous as long as it interacts with the environment.
- Off-policy: that it learns from data that was collected while following a different policy than the one it is currently using

Behind the DDPG algorithm, therefore, we have two processes running in parallel, Q-learning and Policy-learning. In other words, during training, the agent not only learns a way in which to best interpret the goodness of its actions (Q-learning), but also learns in parallel a sequence of actions that seek to maximise the effectiveness of the same actions (Policy-Learning).
DDPG maintains four function approximators:

- Actor: Takes an observation and outputs an action.
- Target actor: A copy of the actor network that is used to stabilize training.

- Critic: Takes an observation and an action as inputs and outputs the expected long-term reward.

- Target critic: A copy of the critic network that is used to stabilize training.

The actor and critic networks are trained together. The actor is trained to maximise the expected long-term reward, as predicted by the critic. The critic is trained to accurately predict the expected long-term reward for each action-state pair.
The target actor and critic networks are used to stabilise training. They are updated less frequently than the actor and critic networks, and they are used to generate the targets for the actor and critic networks. This helps to prevent the actor and critic networks from over-fitting to the training data.
In other words, the target actor and critic networks are used to provide more stable targets for the actor and critic networks to learn from. This helps to improve the performance of the algorithm [47] [48].

The Q-Learning is the process by which the NN of Reinforcement Learning learns to estimate the Q-function, which is the mathematical tool with which the RL understands whether the action taken in a given state was a good action or not. Mathematically speaking, the Q-function is a procedure of minimising the loss of the mean-squared Bellman error (MSBE) function by means of a stochastic gradient descent algorithm, as already anticipated in the formula 3.6, from which the more complete formula can be written:

$$L(\pi, D) = \mathbb{E}_{(s,a,r,s',d)\sim D}\left[\left(Q_\pi(s,a) - (r + \gamma(1-d)\max_{a'} Q_\pi(s',a'))\right)^2\right] \tag{3.7}$$

Policy evaluation is that process by which the agent learns to follow a set of commands aimed at maximising the expected future reward, or in mathematical words, to find that policy π such that it maximises $Q_\pi(s,a)$. The algorithm for solving this problem is the gradient ascend, the mathematical formulation of which is as follows:

$$\max_\theta \mathop{\mathbb{E}}_{s\sim D}[Q_\pi(s, \mu_\theta(s))] \tag{3.8}$$

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Table 3.1: Pseudocode for DDPG algorithm [48].

3.3 Implementation

Using the Mathworks toolbox on Deep Reinforcement Learning, an agent can be defined in the MATLAB environment. The environment can be defined either in MATLAb or in Simulink. Since in our case the agent has to interact with a LapTime Simulator as well as with a vehicle model both defined in Simulink, it is possible

to select a Simulink model as the environment and implement the rlAgent block in the Simulink library (figure 3.3), thus allowing the agent to interact with the generated Simulink model. Owning the vehicle model in the Simulink environment, it was decided to adopt the latter implementation for ease of operation.

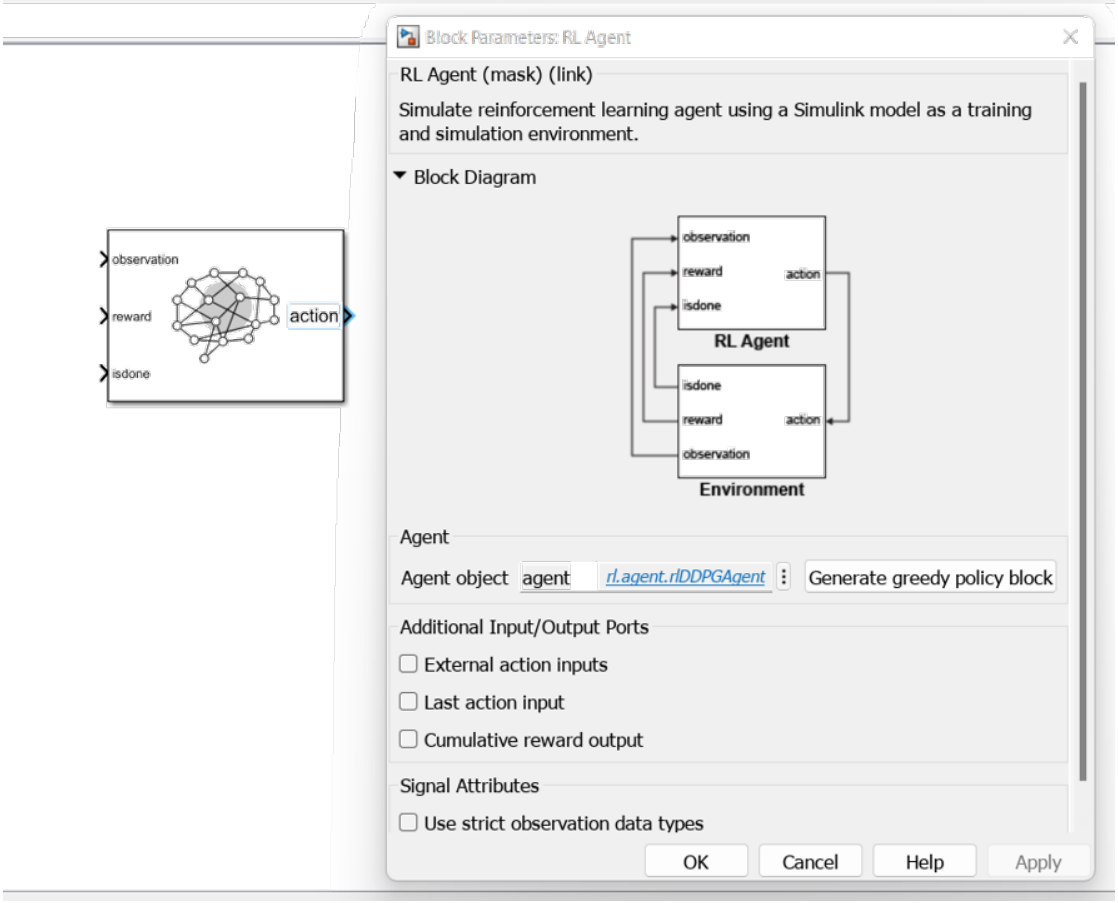

Figure 3.3: RL Agent block from DRL Toolbox of Mathworks.

MATLAB

First, the environment with which the agent is to interact is defined. In particular, it is necessary to define the position of the block within the Simulink environment. Furthermore, fundamental to the success of this thesis project is the definition of the Reset Function via a global function. This is a function that is executed at the end of each episode and is crucial to the success of this thesis project. In fact,

during RL training, it is not possible to access or modify the Matlab workspace. Therefore, if you want to modify an element at the end of an episode, you first have to pre-define it, whereas to modify it without acting in the workspace, you either use a Matlab function defined in Simulink or rely on ResetFcn. Both approaches have been adopted, for more details please refer to the chapter 4.

Listing 3.1: Environmental definition

```
%-------------------------------------------------------------------
%% Initializing the RL agent block in simulink

env = rlSimulinkEnv("Vehicle_model_2023","Vehicle_model_2023/
    Model/Lateral_Longitudinal_Controller/Reference_path/BUFFER/
    RL Agent",...
     obsInfo,actInfo);
%-------------------------------------------------------------------
%% Reset Function  as a local function in oder to act into the
% workspace at the end of each episode:

env.ResetFcn = @(in)localResetFcn(in,Buffer_Data,Manoeuvre);
```

Critic Network

As already discussed, there are two distinct networks in DDPG architectures, the actor and the critic. The critic's network in particular is responsible for estimating the Q-function, which indicates the quality of a given action in a specific state. This network takes as input both the current state and the current action and estimates the value of the Q-function. Since it is designed to estimate the Q function (the action value function defined in eq. 3.7), which depends on both the current state and the current action it is needed two distinct network, one that observes the state and the other that observe the action, the 'state path' and the 'action path' respectively:

- Action Path: This path receives the current action (the decision made by the agent) as input and is responsible for processing it. In other words, it evaluates how the current action affects the Q-function.
- State Path: This path receives the current state of the environment as input.

It is responsible for processing the current state and evaluates how the current state affects the Q-function.

Subsequently, the outputs of these two paths are combined to obtain the final estimate of the Q function. This estimate is used to guide the agent's training in the environment.
The critic's network in DDPG uses both an action path and a state path to capture the complex dynamics between actions and states in the environment in order to correctly estimate the Q function and guide the agent's learning [13].
The critic network chosen is made of two fully connected layers with a ReLU layer in between, this for each path. Then the common path is made by a single ReLU layer, which is an activation function, meaning that will set all its input below the threshold into a null value. Each Fully Connected layer has *numNeurons* set equal to 200.

Listing 3.2: Critic Network

```
%% 01. Creating the critic network

%-------------------------------------------------------------------------
% Takes an observation and an action as inputs and outputs the
    expected long term reward.
%-------------------------------------------------------------------------
% Defined with two path, one for state estimation and one for
    the action.
% The two parts have a path in commond.

statePath = [
     featureInputLayer(obsInfo.Dimension(1),Name="netObsIn")
     fullyConnectedLayer(numNeurons,Name="FCs1")
     reluLayer(Name="RUs1")
     fullyConnectedLayer(numNeurons,Name="FCs2")];

actionPath = [
     featureInputLayer(actInfo.Dimension(1),Name="netActIn")
     fullyConnectedLayer(numNeurons/2,Name="FCs3")
     reluLayer(Name="RUs2")
```

```
19      fullyConnectedLayer(numNeurons,Name="FCs4")];
20
21  commonPath = [
22      additionLayer(2,Name="add")
23      reluLayer(Name="RUs3")
24      fullyConnectedLayer(1,Name="CriticOutput")];
25
26  %-------------------------------------------------------------------------
27  % Each path created now has to be linke to create the NN
28
29  criticNetwork = layerGraph();
30  criticNetwork = addLayers(criticNetwork,statePath);
31  criticNetwork = addLayers(criticNetwork,actionPath);
32  criticNetwork = addLayers(criticNetwork,commonPath);
33
34  % specifying which parts of the NN needs to be connected
35  criticNetwork = connectLayers(criticNetwork, ...
36      "FCs2", ...
37      "add/in1");
38  criticNetwork = connectLayers(criticNetwork, ...
39      "FCs4", ...
40      "add/in2");
```

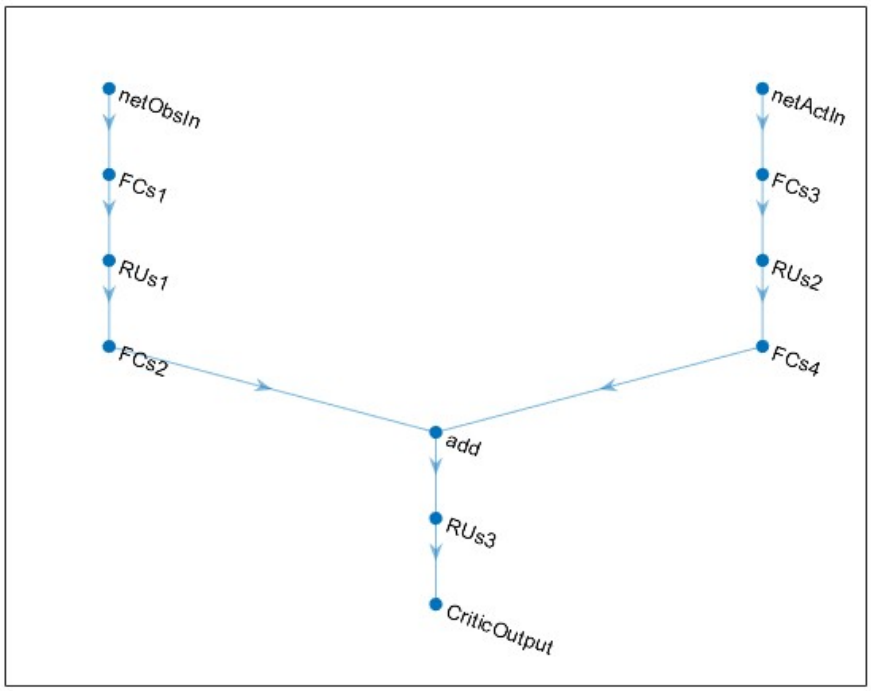

Figure 3.4: Critic Network.

Actor Network

The actor network is responsible for learning and determining the agent's policy, i.e. which action should be chosen in a given state of the environment. The actor network accepts the current state as input and returns the intended action. In practice, it is responsible for the agent's action decision.

Differently from the critic network, the actor network will have only one path since its purpose is to determine policy, i.e. to choose actions. In practice, using a single actor network can simplify the agent architecture and reduce the computational complexity. The agent learns directly from the environment how to choose optimal actions to maximise a reward function. The actor network is trained via gradient back-propagation to optimise the policy in order to maximise the value function Q estimated by the critical network.

The definition of the actor's network follows that of the critic, what changes is its

architecture. In fact it is defined by three Fully Connected Layers, each with 200 neurons as in the critic's network, followed by the ReLU activation function, except for the last layer where a Hyperbolic Tangent activation function is preferred. This choice is due to the fact that tanh is not only limited to values of 0 and 1 but also allows for values between -1 and 1, which is very suitable when actions can be both positive and negative or when more flexibility in the choice of actions is required. the code adopted for its formualtion is here presented:

Listing 3.3: Actor Network

```
%% 02. Creating the actor network

%---------------------------------------------------------------
% Takes an observation and outputs an action.
%---------------------------------------------------------------
% Actor network definition

actorNetwork = [
    featureInputLayer(obsInfo.Dimension(1),'Name','observation')
    fullyConnectedLayer(numNeurons,Name="FCa1")
    reluLayer(Name="RUa1")
    fullyConnectedLayer(numNeurons,Name="FCa2")
    reluLayer(Name="RUa2")
    fullyConnectedLayer(actInfo.Dimension(1),Name="FCa4")
    tanhLayer(Name="Tanha1")
    scalingLayer('Name','SCa1')
    ];

actorNetwork = dlnetwork(actorNetwork);
```

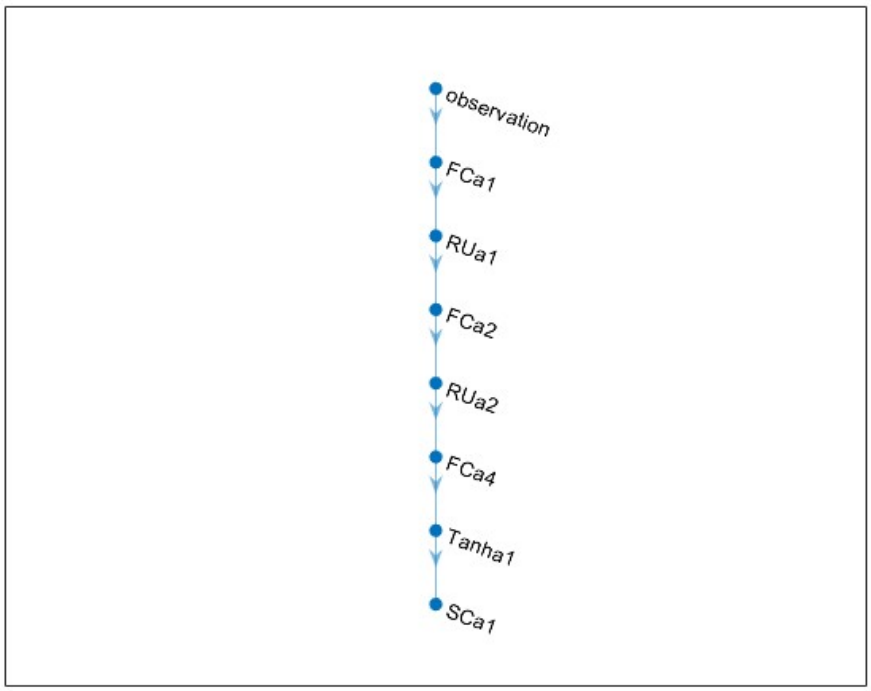

Figure 3.5: Actor Network.

Simulink

Once the agent has been defined, it is implemented in two different models (figure 3.6 and 3.7). In the first application, the agent receives as observers the centreline trajectory of a given circuit, the action it performs (defined as a lateral deviation from the centreline of the circuit) and the lap time information. The reward function adopted instead is based on the same information with the addition of the trajectory in the inertial reference system. A memory block is used to allow the signals generated by the agent to have an initial condition on the first iteration and to avoid the creation of algebraic loops.

With regard to the application of the agent in the buffer system, on the other hand, the vehicle states (coordinate x and y in the inertial reference system, yaw angle psi, longitudinal and lateral accelerations ax and ay and steering action δ_f) were

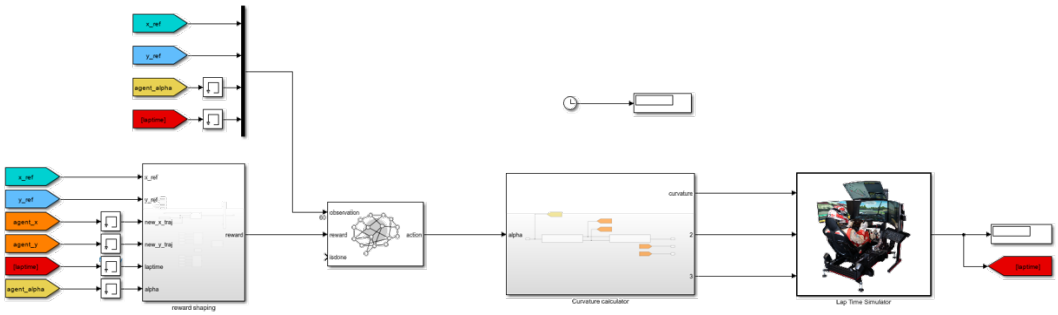

Figure 3.6: Simulink model created for lap time simulations.

set as observations. These signals refer both to the current state of the vehicle and to previous laps, thanks to a data buffer that is consulted according to the curvilinear co-ordinate on the centreline trajectory, referring to a moving window integral with the vehicle.

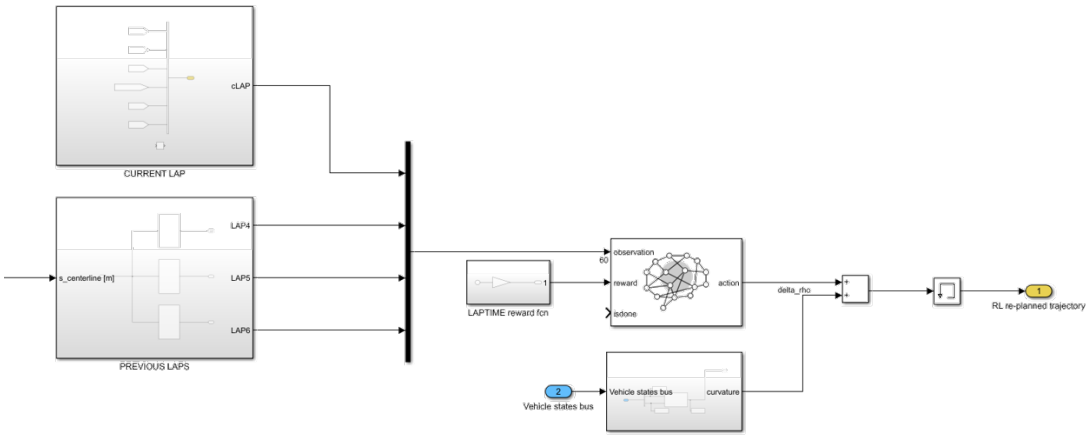

Figure 3.7: Simulink model created for lap time simulations.

Chapter 4

Trajectory generation for QSS point mass vehicle model

4.1 Tool-chain introduction

As a first playground for defined artificial intelligence, it was chosen to test trajectory generation on an open source lap time simulator. OpenLAP is a lap time simulator designed by Cranfield University. Its features include simplicity of implementation, speed of execution, good reliability and the ability to have a large amount of signals generated.

This is a simple MATLAB code saved as an .m file, it is executable on later versions of MATLAB 2015. It consists of three scripts, OpenVEHICLE, OpenTRACK and OpenLAP. The first, as the names suggest, allows the vehicle parameters to be defined and generates from these the g-g diagram used during the lap time simulation. The second script allows you to define the trajectory to be followed by the vehicle during the lap time. In fact, OpenLAP is a lap time simulator on a fixed trajectory, i.e. it is not capable of calculating the optimal trajectory, but is limited to simulating the vehicle along a given trajectory. Therefore, it should

be emphasised that the trajectory is an input to the system. In the application studied, in fact, the input is generated by the agent. Finally, the third script allows true lap-time simulation using the selected vehicle model and the given trajectory as input.

Both the vehicle and the trajectory are perfectly customizable from csv files. In fact, the code of OpenVEHICLE as well as that of OpenTRACK read .csv files via the `xlsmread` function of MATLAB, extract the data and save them in structures that will later be used by OpenLAP. In the proposed work, therefore, a specific circuit was selected, namely the first sector of the Berlin track used in the Formula E calendar 2018 for its shortness yet its demanding vehicle dynamic performance, as well as the model of a single-seated similar to an SAE formula was implemented.

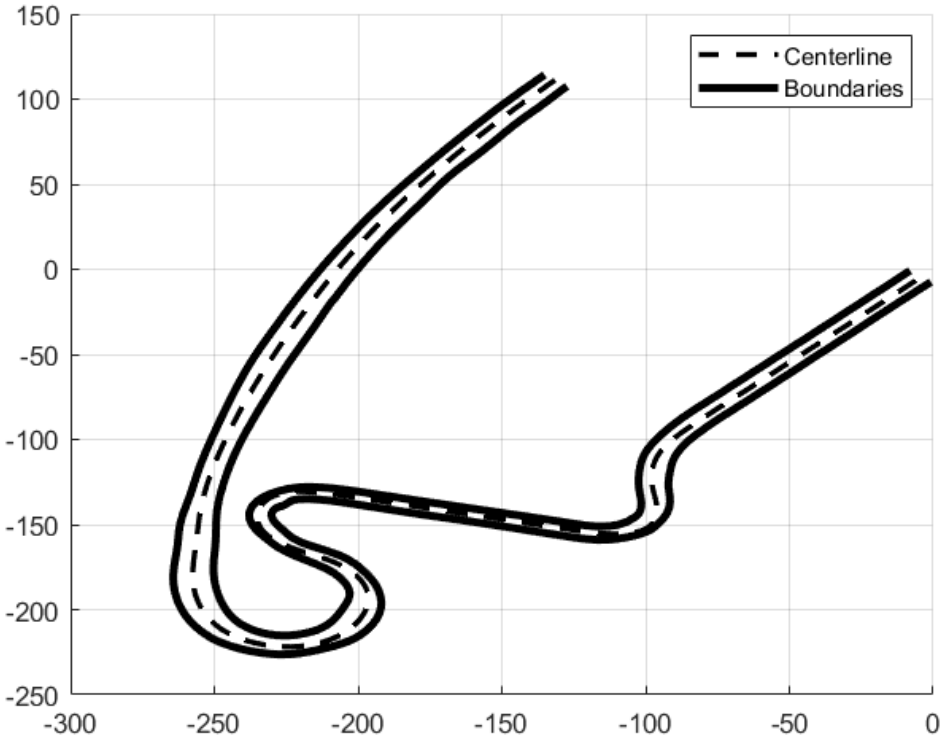

Figure 4.1: First sector of Berlin Formula E 2018 track, used in this project.

4.1.1 Vehicle model adopted

The vehicle model adopted in OpenLAP is a Quasi-Steady-State Point Mass vehicle model and adopts the procedure introduced in section 2.2.1. However instead of computing the vehicle state-system at each step, an algorithm to estimate the velocity profile is used and presented in section 4.2.
The generation of the car model is made from an Excel file. Here all the parameters can be defined and modified. Notable, the trajectory generation is performed considering that the vehicle is kept the same along the training. Therefore the vehicle model is created as a Matlab structure only once and is done prior to the lap time simulation.

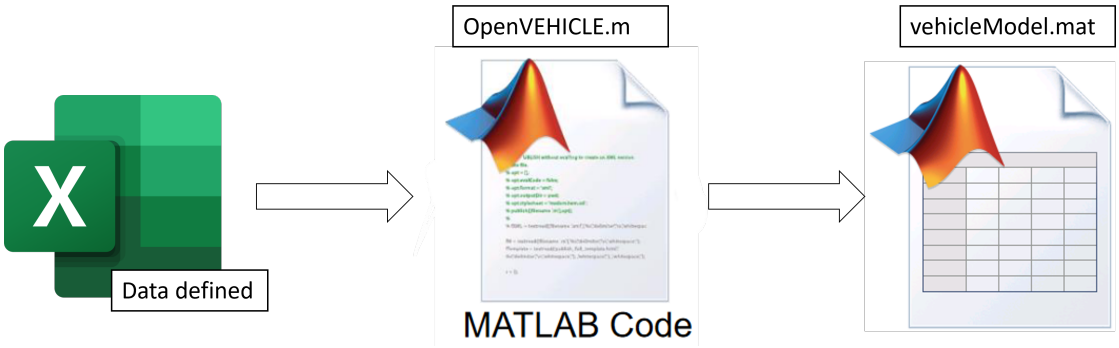

Figure 4.2: Workflow of OpenVEHICLE script.

The vehicle model selected resembles a Formula SAE car, whose parameter can be summarised in table 4.1.

4.1.2 Trajectory generation

The trajectory generation follows the same principle of the vehicle model. However, at each episode the trajectory changes since it is generated by the agent, the file needs to be modified at the end of each simulation. This process will be deeply explained in the section 4.3.
Although the output of the OpenTRACK code is a structure that contains information about the circuit, it is good to keep in mind that the trajectory is defined within it by the radius of curvature ρ.

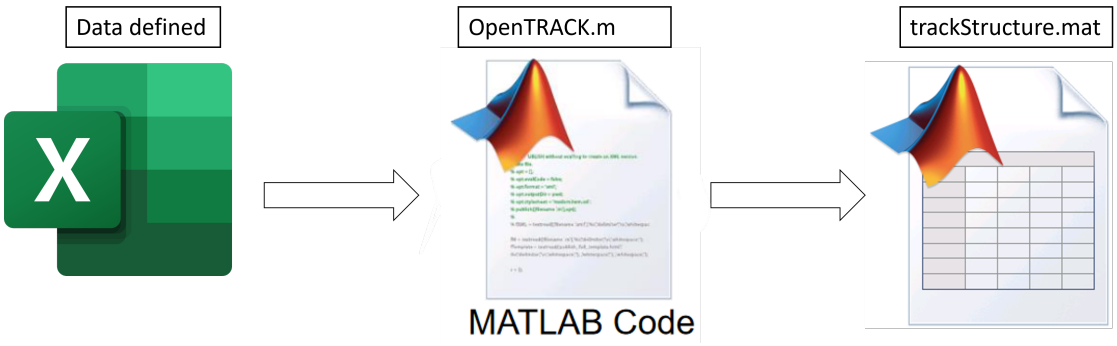

Figure 4.3: Workflow of OpenTRACK script.

4.2 OpenLAP Lap time simulation

Algorithm

Basically this Lap time simulator has as its goal to estimate the velocity profile of the vehicle and is done by 4 steps: Apex findings and speed values calculation; Acceleration; Deceleration; Braking points.

The apex is defined where the vehicle has a local minima of the speed. From here, it is assumed that the vehicle will accelerate and before that point the vehicle will brake. The velocity profile therefore will, follows the acceleration velocity profile until the intersection point with the following braking curve is met, and afterwards the vehicle velocity will follow the braking curve.

The algorithm proceeds as follows: at each point the maximum lateral velocity is independently estimated by pure lateral motion. However the velocity computed could lead to the situation where the drag force over exceed the longitudinal force created by the car. Physically it means that the vehicle should generate some longitudinal force which will reduce the lateral acceleration and, therefore, the velocity at which the car is turning. This is done by reverse computing the actual velocity.

From the assumed velocity (calculated from pure lateral acceleration), the tyre forces are computed and from these the lateral acceleration needed to develop such forces is estimated. By making a comparison with the friction ellipse (equal to the g-g map), the *ay* actually available is read and if the acceleration needed overweight the acceleration available, the velocity is slightly reduced. This process

is reiterated until the latter condition is true.
Once the velocity at each apex is defined, several points along the circuits are defined by the apex, and from there the all lap velocity profile is deduced following the acceleration and deceleration curves.

4.3 Implementation

OpenLAP was programmed as a Matlab file, so implementing it on Simulink required a lot of adaptation and code reconstruction. In particular, several Matlab functions were written and implemented as Matlab function in the Simulink environment.
implementation procedure of OpenLAP in the Simulink environment follows the standard procedure done by the original author. In fact, first useful parameters are defined, such as the vehicle model and simulation parameters, such as sampling time and pre-allocation of variables for code stability.
Next, the actual execution of the simulation can take place. Starting from the assumption that the RL agent generates a trajectory (the explanation of which is detailed in the section 4.3), this is passed to a function that modifies the csv file, allowing OpenTRACK to generate the trajectory structure. To do this, a function was created ad hoc, the code for which is given here:

Listing 4.1: csvGeneratefunction

```
function [baseFileName] = generateCSVfunction(curvature,agent_x
    ,agent_y)
%-------------------------------------------------------------------
% MATLAB function to take an array, write it on a table and
    save into .xlsx
%-------------------------------------------------------------------

% It is needed to usa a function instead of a .m script due to
    code
% generation requirements

%-------------------------------------------------------------------
     warning("off","all") ;
```

```
11      addpath("csvOfTrajectoriesForOpenLAP")
12
13      % Directory definition
14      folder       = "csvOfTrajectoriesForOpenLAP" ;
15
16      % Assuring that the file is being closed, otherwise the
        writing is
17      % denied
18      fclose("all") ;
19
20      % Creation of the directory if not existing
21      if ~exist(folder, 'dir')
22          mkdir(folder);
23      end
24
25  %-------------------------------------------------------------------
26  % File and arrays creation - NOTE: Check the name of the xlsx
        created
27  %-------------------------------------------------------------------
28
29      % File creation
30      baseFileName         = "Agent_sector_2_Flip_0_traj_ltpl_cl";
31      fileExtension        = ".xlsx";
32      fullFileName         = fullfile(folder,baseFileName+
        fileExtension);
33
34  %-------------------------------------------------------------------
35  % Column modification - 3 Columns have to be modified: rho, x,
        y
36  %-------------------------------------------------------------------
37
38      % Write curvature to the second sheet in the third column,
        starting from the second row
39      corner_sheet = 2;
40      corner_range = 'C2';
41      corner = 1./abs(curvature);
```

```
42      writematrix(corner, fullFileName, 'Sheet',corner_sheet,'
    Range',corner_range);
43
44      % Write agent_x to the seventh sheet, starting from the
    second row
45      agent_x_sheet = 7;
46      agent_x_range = 'A2';
47      writematrix(agent_x, fullFileName, 'Sheet',agent_x_sheet,'
    Range',agent_x_range);
48
49      % Write agent_x to the seventh sheet, starting from the
    second row
50      agent_y_sheet = 7;
51      agent_y_range = 'B2';
52      writematrix(agent_y, fullFileName, 'Sheet',agent_y_sheet,'
    Range',agent_y_range);
53
54      % Close to avoid multiple file running
55      fclose("all");
56  end
```

The csv used by OpenTRACK must have a precise formatting. In sheet 1,3,4,5,6,7 some info over the track are given are given, meanwhile in sheet 2 the trajectory to e followed must be declared in three columns: left/right/straight information, the section length and the corner radius in meters. Please note that the csv file requires the corner of the curve radius, not the curvature. For this reason the calculation at line 41 is performed. An example for the first meters of the Berlin Formula E 218 track is reported in the table 4.2.

Type	SectionLength[m]	CornerRadius[m]
Right	2.998027318	22.7281369
Right	2.998047955	2.44869273
Right	2.998074969	69803.8729
Right	2.998107933	72.6767477
Right	2.998147286	5.30194156
Right	2.998192619	19.8624412
Right	2.99824421	17.6776915
Right	2.998302161	2.56749936

Table 4.2: OpenTRACK csv example for initial part of Berlin Formula E 2018 track

Afterwards, it is possible to have the codes used to execute OpenTRACK and OpenVEHICLE. However, normally the two codes communicate with each other by creating variables which are saved in structures which are then called up by the next code. In Simulink, this operation would not only be laborious, as it would involve the massive use of data buses, but is not easily combined with the limitations imposed during Reinforcement Learning. In fact, during this procedure, due to structural limits imposed by Mathworks, the workspace within which the data is saved is not accessible [49]. Therefore, if you wish to add or delete a variable during agent training, you must declare a local Matlab function. The latter in fact allows a variable to be declared and used as desired without affecting the workspace, which is instead composed of global variables [50]. For this reason, it was decided to communicate the OpenTRACK script with the OpenLAP script using local functions.

To recapitulate, the OpenTRACK code has been introduced into a function which reads the csv file, generates the track data including the trajectory to be followed and as output has an array of cell elements, where each element corresponds to a field of the structure that the original script predicted. This array of cells is passed as input to an OpenLAP pre-launching script, which extracts the data from the array and converts it into the format that the original OpenLAP script envisaged. Finally, OpenLAP, adapted as a local function named $functionOpenLAP\_GR$, is executed using the data read from the array. The codes of the following functions

are proposed below, avoiding the inclusion of the OpenTRACK and OpenLAP scripts, not only because they are open source and therefore available to the reader, but since they are codes of 902 lines the former and 1138 the latter, they would unnecessarily prolong the text without bringing any particular additional information.

Listing 4.2: LaunchOpenTRACK

```
function trTableData = LaunchOpenTRACK(baseFileName)
%-------------------------------------------------------------------------
% OpenTRACK script adapted as a function for Simulink environment.
%-------------------------------------------------------------------------

    % This fucntion creates the necessary files to simulate a proper lap
    % time into OpenLAP.

        % xlsx Name -> this file -> cell(data for OpenLAP)

    % The generation is based on the reading of a xlsx
    % file, in which the trajectory is defined as a distance travelled and
    % curvature ("Shape sheet" - Sheet 2 of the xlsx file)

    % This script takes as an input the file name of the xlsx file (without
    % the extension!) and gives as an output a cell. Here are included
    % informations about the track and the trajectory to be simulated.

%-------------------------------------------------------------------------
% OpenTRACK script
%-------------------------------------------------------------------------


```

```
[...]

%-------------------------------------------------------------------
% Output Definition
%-------------------------------------------------------------------

      trTableData = cell(1, numel(fieldnames(tr)))';

    % Get the field names of the structure
    fieldNames = fieldnames(tr);
    trTableData = struct2cell(tr) ;
end
```

Listing 4.3: LaunchOpenLAP

```
function laptime = LaunchOpenLAP(trTableData)
%-------------------------------------------------------------------
% Function which act as the launcher for OpenLAP script for
    Simulink environment.
%-------------------------------------------------------------------
    % This function simulate a laptime for the track and
    trajectory
    % characteristics stored into the cell variable.

    % This script takes as an input the cell containing the
    varibales
    % needed and gives as an output the laptime in seconds.

        % cell(track and trajectory) -> this file -> laptime
%-------------------------------------------------------------------
% Definitions
%-------------------------------------------------------------------

    % Take attention to the index numerations!
    check_savingIndex = false ;
    if check_savingIndex
        type Nomenclature_input_for_functionOpenLAP.txt
```

```
20      end
21
22      % Extracting infomration: cell -> arrays
23      tr_x                    = cell2mat(trTableData(1)) ;
24      tr_X                    = cell2mat(trTableData(2)) ;
25      tr_r                    = cell2mat(trTableData(3)) ;
26      tr_dx                   = cell2mat(trTableData(4)) ;
27      tr_n                    = cell2mat(trTableData(5)) ;
28      tr_Z                    = cell2mat(trTableData(6)) ;
29      tr_bank                 = cell2mat(trTableData(7)) ;
30      tr_incl                 = cell2mat(trTableData(8)) ;
31      tr_factor_grip          = cell2mat(trTableData(9)) ;
32      tr_sector               = cell2mat(trTableData(10)) ;
33      tr_Y                    = cell2mat(trTableData(11)) ;
34      tr_apex                 = cell2mat(trTableData(12)) ;
35      tr_r_apex               = cell2mat(trTableData(13)) ;
36      tr_info_config          = "Closed" ;
37      tr_info_name            = "Try" ;
38
39  %-------------------------------------------------------------------------
40  % OpenLAP simulation
41  %-------------------------------------------------------------------------
42
43      % Function launch that resamble the OpenLAP script
44      laptime = functionOpenLAP_GR(tr_X,tr_Y,tr_Z,tr_apex,tr_bank
        ,tr_dx,tr_factor_grip,tr_incl,tr_info_config,tr_n,tr_r,
        tr_r_apex,tr_sector,tr_info_name,tr_x) ;
45      fclose('all') ;
46  end
```

RL Implementation

In the context explained, the RL is the trajectory generator. These are entered into the csv file and then read and passed to the LaunchOpenTRACK function. The agent's observations are the global co-ordinates, the action it performs and

the lap time due to the action performed. The agent is set to generate trajectories, in particular it generates a lateral deviation from the centreline normal. Therefore what the agents learns is how to deviate from the centreline in order to reduce the lap-time. Piratically, the output of the agent is a vector containing a scalar value which has to be add to the centreline reference trajectory.

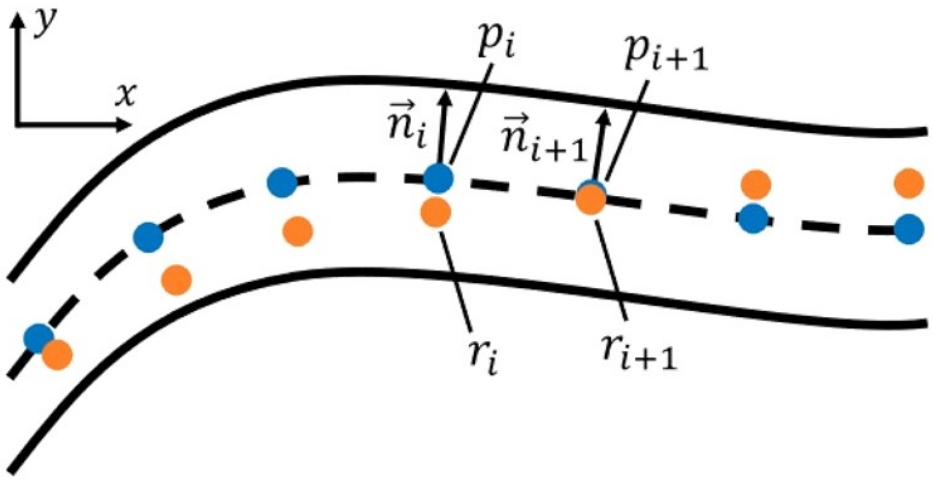

Figure 4.4: Lateral deviation following the centreline normal.

This approach presents a novelty compared to the literature. In fact, the path-planning process tends to be performed by optimisers that require considerable computational effort. RL was mainly used as path-tracking. Or if it has been used as path-planning, it is because it was involved in an end-to-end autonomous approach, i.e. the agent learns from the video input and learns to generate control signals (make reference to the literature review proposed in section 2).

The proposed work therefore proposes the idea of presenting the RL as a trajectory optimiser that is able based on lap-time observations to adapt to the vehicle and track context. Indeed, as the DDPG is a model-free algorithm, it should learn an optimisation strategy that only considers the effects of its actions on the environment, without taking into account the nature of the model it is acting on. This approach could be useful in the racing context, where the agent should adapt in real-time to modifications in the vehicle's performance and the ever-changing

track conditions.

Fundamental to the correct training of the agent is a properly defined reward function for the proposed porblem [13]. In the case studied, it was decided to set a lap time for an optimised trajectory, obtained by executing the optimisation process proposed in the reference [12].
The proposed reward function is based on two main signals: a time delta and the action of the agent. Both of these quantities have an associated weight that must be properly defined. In the proposed case, the chosen values are the results of a trial-and-error procedure.

$$\Delta_{lap-time} = t^{optimiser} - t_{i-1}^{agent} \tag{4.1}$$

Thus, one of the signals on which reward function is based is the time difference between this optimised trajectory and that of the agent. This provides an incentive for the agent to learn this trajectory in the first phase, and then later explore other solutions. The lateral deviation from the centreline α, defined as the agent's action in the antecedent step. To encourage the agent to further reduce the lap-time, the H element is introduced, which becomes equal to 1 if the $\Delta_{lap-time}$ is below $1seconds$. Finally, the reward function used can be formulated:

$$reward = -(100\Delta_{lap-time} + 500u^2) \times (-1e^{-3}) + 2H \tag{4.2}$$

4.4 Critical analysis

The results of the training procedure explained in this chapter are pressed in the figure 4.5. Here it can be seen that despite the high number of episodes (corresponding to approximately 24 hours of uninterrupted calculation), the reward although has an upward trend, it does not reach satisfactory values, which physically translates into a difficulty on the part of the agent in generating competitive trajectories.
The cause of this can be associated with several reasons, including:

- The heaviness of the process of reading and writing from the disk.

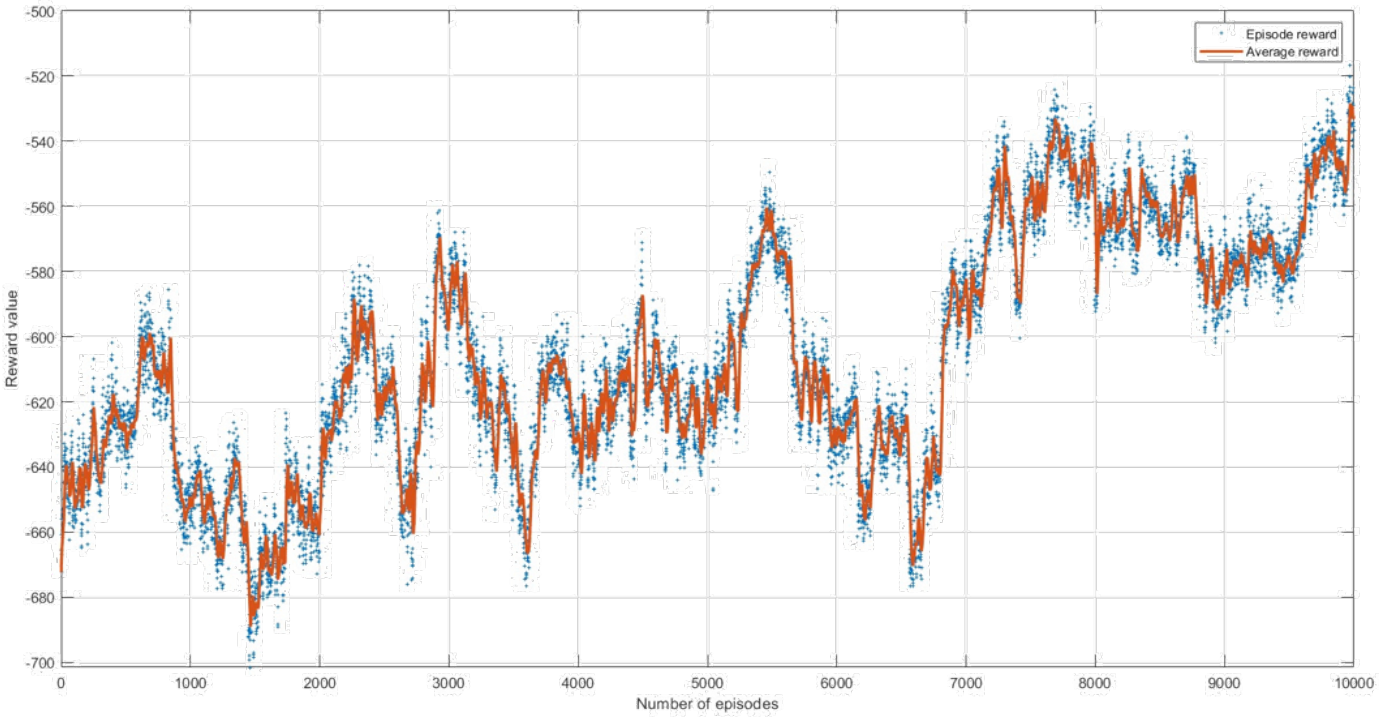

Figure 4.5: Training window of the RL.

Although the toolchain works, necessarily having to go through the reading and writing of an excel file is not the most efficient process possible. In fact, among computer actions, the process of reading a file, editing it and saving it is among the most demanding computationally speaking. This implies that the simulation is slow and inefficient, and these are not the optimal condition for a proper neural network training.

- Mismatch between the demands of the plant and the agent.

  A weakness of OpenLAP is that it requires the trajectory to be fully defined before simulation takes place. In the case studied, only the first sector was evaluated. However, the longer the sector to be analysed, the greater the length of the vector to be generated by the RL. As can easily be guessed, the larger the size of the action to be performed by the agent, the more difficult it will be to converge to a solution. This problem is called the curse

of dimensionality [51]. In the case studied, therefore, it is difficult to reach a compromise, since on the one hand OpenLAP requires a suitably long sector of the track to be analysed in order to have sufficient information, while on the other hand the agent would require a reduction in the array it has to generate, which would imply a smaller section of the circuit. This effect of decompensation slows down the training process considerably, not allowing the network to converge to an optimal result in a time that is not exaggeratedly long.

- Need for a more realistic model.

  Although OpenLAP is demonstrably a good lap time simulator, for the needs of the proposed thesis it may have limitations. In fact, since the idea is to be able to allow online training based on data from previous laps, it is consequently necessary to have a vehicle model that is as representative as possible. In fact, since the training must be based on the signals coming out of the plant, the more reliable the plant is, the more satisfactory the training of the network, and its ability to perform an optimal control action, will be. Hence the need to change the vehicle model and switch to a more representative and comprehensive system than a lap time simulator.

Name	Formula 1	-
Type	Open Wheel	-
Total Mass	1200	kg
Front Mass Distribution	53	%
Wheelbase	3000	mm
Steering Rack Ratio	10	-
Lift Coefficient CL	-1	-
Drag Coefficient CD	-1.246	-
CL Scale Multiplier	1	-
CD Scale Multiplier	1	-
Front Aero Distribution	50	%
Frontal Area	1	m^2
Air Density	1.2041	kg/m^3
Disc Outer Diameter	250	mm
Pad Height	40	mm
Pad Friction Coefficient	0.45	-
Caliper Number of Pistons	6	-
Caliper Piston Diameter	40	mm
Master Cylinder Piston Diameter	25	mm
Pedal Ratio	4	-
Grip Factor Multiplier	1.2	-
Tyre Radius	300	mm
Rolling Resistance	-0.001	-
Longitudinal Friction Coefficient	1	-
Longitudinal Friction Load Rating	300	kg
Longitudinal Friction Sensitivity	0.0001	1/N
Lateral Friction Coefficient	1	-
Lateral Friction Load Rating	300	kg
Lateral Friction Sensitivity	0.0001	1/N
Front Cornering Stiffness	1200	N/deg
Rear Cornering Stiffness	1200	N/deg
Power Factor Multiplier	1	-
Thermal Efficiency	0.35	-
Fuel Lower Heating Value	4.72E+07	J/kg
Drive Type	RWD	-
Gear Shift Time	0.05	s
Primary Gear Efficiency	0.98	-
Final Gear Efficiency	0.9	-
Gearbox Efficiency	0.98	-
Primary Gear Reduction	0.98	-
Final Gear Reduction	1	-
1st Gear Ratio	1	-

Table 4.1: Vehicle parameters adopted.

Chapter 5

Path re-planning

In this chapter the second part of the thesis is explained. It concerns the developing of a tool-chain capable of store vehicle's signals generated online into a buffer which is used as a data provider for the Reinforcement Learning agent during its training. In particular, emphasis was placed on the implementation of buffers that allow online consultation and updating at the end of each episode.

5.1 Tool-chain for data retrieval

Given the difficulties of Reinforcement Learning in generating trajectories for entire sections of the circuit, it was decided to implement it in an contest where it only generates the instantaneous trajectory that the vehicle will follow. A manoeuvre is defined, in the case studied a 90 degree turn to the left was chosen. The agent trains by taking into account the data collated from previous turns, consulted by taking into account the position of the vehicle and a moving window that moves with it. The input used to read the data is the curvilinear co-ordinate along the centreline trajectory, to use a distance-independent reference. A 8 DOF vehicle model is used as a high-fidelity plant for higher dynamic accuracy of the system. Two controllers are used to make the plant follow the trajectory generated by the agent: a PID for longitudinal dynamics and an FF-FB for lateral dynamics. At each end of an episode, set equal to the length of the manoeuvre, the data buffer is updated by saving the last recorded data and deleting the oldest data.

In the first phase of training the buffer will contain information on manoeuvres that are not quite optimal, but as the agent improves his trajectory, the buffer will consequently store more and more meaningful manoeuvre data that will facilitate network learning.

5.1.1 Vehicle model

The vehicle model presented in Chapter 2 was implemented. It consists of a 8 DoF model, corresponding to longitudinal and lateral coordinate, yaw and roll motion, and the dynamics of the four wheels. The equation have been presented in section 2.2.3 meanwhile here its implementation is presented.

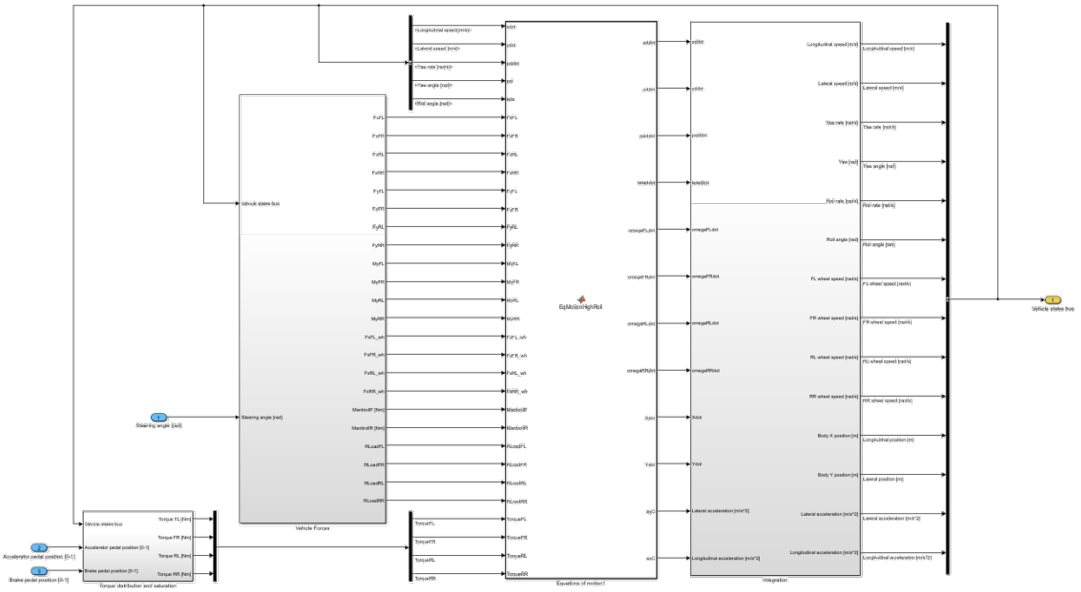

Figure 5.1: 8 DoF Simulink implementation.

This model takes as inputs the acceleration7braking command, which is generated from the longitudinal dynamics - PID and the steering angle, coming as an output from the lateral FF-FB controller. The output of the model are the vehicle model state bus, containing information over the longitudinal speed, lateral speed, yaw rate, yaw angle, roll rate, roll angle, the four wheels speeds, longitudinal and lateral positions, lateral and longitudinal accelerations. The vehicle model is implemented as a MATLAB function containing the equations presented in section 2.2.3.

5.1.2 Calculation of s

The information over which the buffer is read is though the curvilinear coordinate, also referred to as s. This information has to be estimated from the vehicle states following the equation 5.1. Considering a generic point moving along a trajectory and a reference point, the system in figure 5.2 can be draw. The value of $\dot{s}$ can be derived. From this information, the curvilinear coordinate s can be obtained trough integration [14].

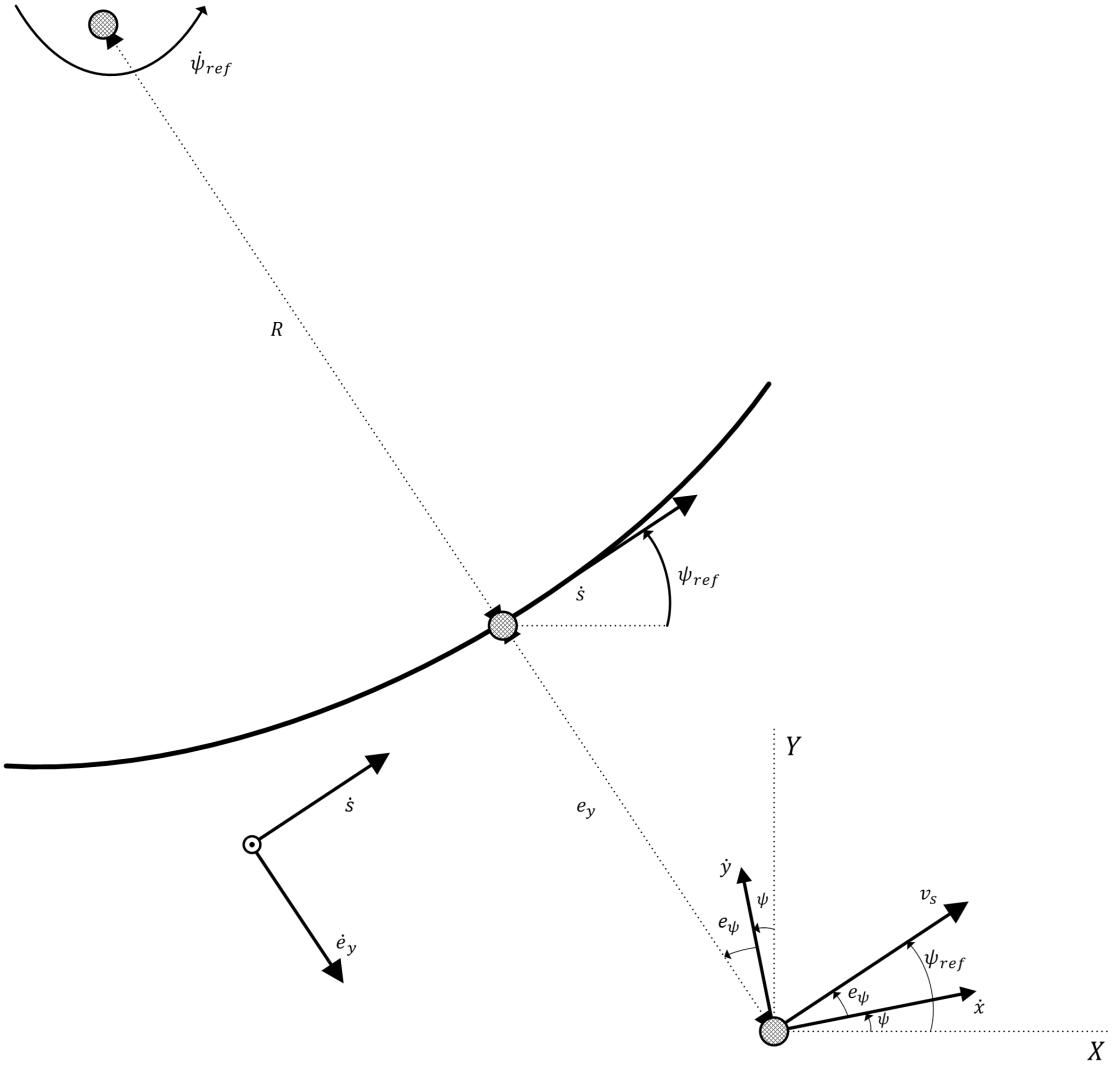

Figure 5.2: $\dot{s}$ model representation.

$$\dot{s} = \frac{1}{1 + k_{ref} e_y} \left(\dot{x} \cos\left(e_\psi\right) + \dot{y} \sin\left(e_\psi\right) \right) \tag{5.1}$$

The Simulink implementation of the system described is showed in figure 5.3, where in the left block the calculation of $\dot{s}$ is performed according to formula 5.1, meanwhile the right block performs the integration, having as initial condition the value of the curvilinear coordinate at the previous step.

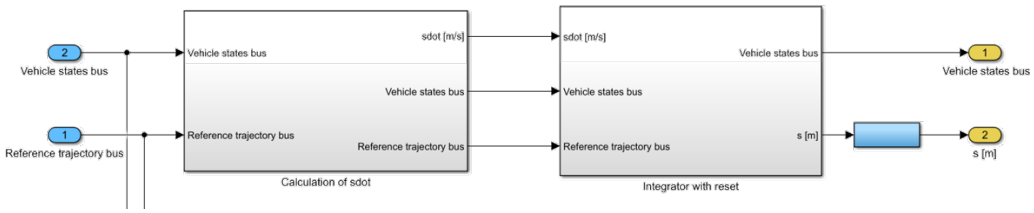

Figure 5.3: s Simulink implementation.

5.1.3 Controllers

Two controllers are adopted to follow the trajectory generated by the agent, one for longitudinal dynamics and the other for lateral dynamics. The longitudinal dynamics is controlled by a PI, while the lateral dynamics is followed by an FF-FB controller. The PI has as input the error on the longitudinal velocity, i.e. the difference between the current vehicle speed and the reference speed of the manoeuvre, and has as output the torque demand. The formulation of the PI in continuous time is presented in the formula 5.2. The Simulink implementation is presented in figure 5.4.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau \tag{5.2}$$

The FF-FB controller serves as a lateral control system. It achieves overall steering control through a combination of two key components: feed-forward and feedback. The implementation of controllers into Simulink environment is showed in picture 5.5.

The feed-forward component determines the required steering angle for steady-state turn negotiation. It is calculated by combining the kinematic steering angle,

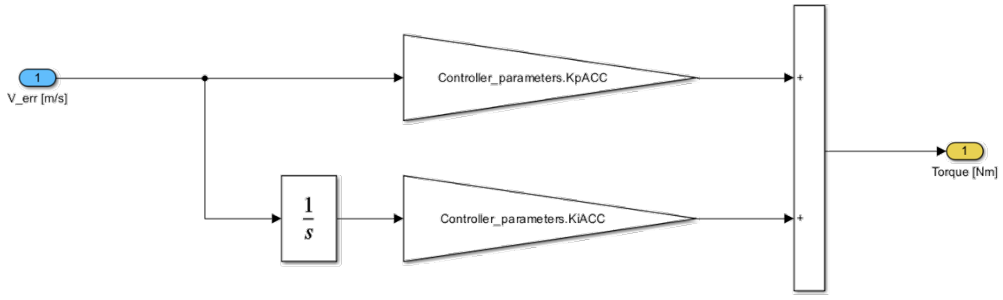

Figure 5.4: PI controller used for longitudinal controller [14].

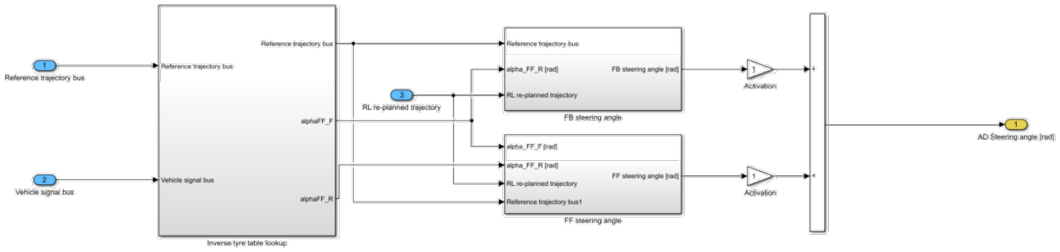

Figure 5.5: Simulink implementation of the FF-FB controller [14].

computed using the Ackermann formula for a single-track model, with the slip angles of the front and rear tires. Generating slip angles involves the creation of lookup tables with lateral forces and vertical loads as inputs, yielding slip angles as outputs. On the other hand, the feedback portion of the controller utilises a proportional controller for two error signals: lateral position error (e_y) and heading error (e_ψ). This feedback is derived from the sum of the lateral position error at the vehicle's centre of gravity (e_y) and the projection of the look-ahead distance in the direction of e_ψ.

The controller's tuning parameters consist of the proportional gain (K_{FF-FB}) and the look-ahead distance (x_{LA}). The former adjusts the relative importance of the entire feedback component in the overall control action, while the latter regulates the balance between lateral deviation at the centre of gravity and heading error. Finally the final formulation of the two controllers is shown in equations 5.3 and 5.4 [14].

$$\delta_{FF} = l_c \rho_{planned} + \alpha_{FF,F} - \alpha_{FF,R} \tag{5.3}$$

$$\delta_{FB} = K_{FF-FB} \left[e_y + x_{LA} (e_\psi - (\alpha_{FF,R} + b_c \rho_{planned})) \right] \tag{5.4}$$

With the default setting of the model, using the explained controller (equations 5.2, 5.4 5.3) and the defined vehicle model (2.2.3) it is possible to perform the first manoeuvres, which will be defined as the standard manoeuvre. Assuming that the ideal trajectory is known, the plant will provide the following results (figure 5.6) for a 90 degree manoeuvre to the left can be observed. The code used for running the Simulation is given in listing 5.1, where all the function used are all used for creating structures containing parameters useful for the Simulink model. In Listing 5.1

Listing 5.1: Simulink_model_startup

```
clear
close all
clc
warning('off')

MyPath = pwd;
addpath(genpath([MyPath, '\1_Parameters']));
addpath(genpath([MyPath, '\2_Model']));
addpath(genpath([MyPath, '\3_Results']));

Simulink_mdl = 'Vehicle_model_2023.slx';

global Simulation_parameters Parameters_car Manoeuvre Obstacle
    Controller_parameters Planner_probl_data Buffer_Data

%------------------------------------------------------------------
% Simulation settings
%------------------------------------------------------------------

% Open / Closed loop test
% 1: Open loop              - LUT table planning
% 2: Closed loop            - Online re-planning
```

```
22 Open_Closed_Loop_sw                  = 2;
23
24 % Open loop test
25 % 1: Straight line test
26 % 2: Ramp steer test
27 % 3: Step steer test
28 % 4: Double step steer test
29 % 5: Sinusoidal test
30 Open_loop_steer_input                = 1;
31
32 % Closed loop manoeuvre
33 % 1: Obstacle avoidance
34 % 2: Skidpad
35 % 3: Imola F1 track
36 % 4: McLaren Logo
37 % 5: Top Gear Test Track
38 % 6: Oval Track
39 % 7: 90 deg corner long Track
40 Manoeuvre_closed_loop             = 7;
41
42 % Closed loop path tracking controller
43 % 1: P
44 % 2: FF-FB Stanford
45 % 3: PD with preview
46 % 4: LQR
47 % 5: LQR with preview
48 % 6: MPC linear
49 % 7: MPC linear lateral and longitudinal variable Cstiff
50 Closed_Loop_Steering_Controller    = 2;
51
52 %-------------------------------------------------------------------------
53 % Simulation parameters
54 %-------------------------------------------------------------------------
55
56 % --- Simulation parameters ---
57 Simulation_parameters = Get_simulation_parameters(Simulink_mdl,
       Open_Closed_Loop_sw,Open_loop_steer_input,Manoeuvre_closed_loop,
       Closed_Loop_Steering_Controller);
58
```

```
59 % ---- Vehicle parameters ----
60 Parameters_car          = Car_parameters(MyPath);
61
62 % ---- Manoeuvre parameters ----
63 Manoeuvre               = Get_manoeuvre_parameters(
       Simulation_parameters, Parameters_car);
64
65 % ---- Obstacle parameters ----
66 Obstacle                = Get_obstacle_parameters();
67
68 % ---- Load planner parameters ----
69 Planner_probl_data      = Get_planner_parameters(Manoeuvre,
       Parameters_car, Obstacle);
70
71 % ---- Controllers parameters ----
72 Controller_parameters = Get_Controller_Parameters_Simulink(
       Simulation_parameters, Parameters_car);
73
74 % ---- Buffer data ----
75 Buffer_Data = Get_Buffer_Data();
76
77 %-------------------------------------------------------------------
78 % Run simulation model
79 %-------------------------------------------------------------------
80
81 clearvars -except Simulation_parameters Parameters_car Manoeuvre
       Obstacle Controller_parameters Planner_probl_data Buffer_Data
82
83 % Open the Simulink model
84 % open(Simulation_parameters.Simulink_mdl_name)
85
86 % Run the simulation
87 % tic
88 sim(Simulation_parameters.Simulink_mdl_name)
89 % toc
90 %-------------------------------------------------------------------
91 % Save data
92 %-------------------------------------------------------------------
93
```

```
94 % Savedata
95
96 %------------------------------------------------------------------------
97 % End
98 %------------------------------------------------------------------------
```

5.2 Buffers

A key point in the development of the defined tool-chain is the development of buffers that allow online consultation of data from previous laps. The final objective of the project, of which the work of this thesis is a part, is to allow a path re-planning that takes into account the evolution of the state of the track or vehicle without direct modelling of the environment or plant being involved. Not surprisingly, an RL agent was chosen due to its model-free learning qualities. However, in order for the agent to learn to adapt to dynamic conditions in the racing environment, (like variations in friction coefficients, alterations in tyre temperature) it is necessary for the agent to observe such changes during training. Hence the need to implement subsets of stored data that maintain dynamic information on previous laps and that can be consulted online by the agent. In the next Chapter (6) the code and implementation of such buffers will be discussed.

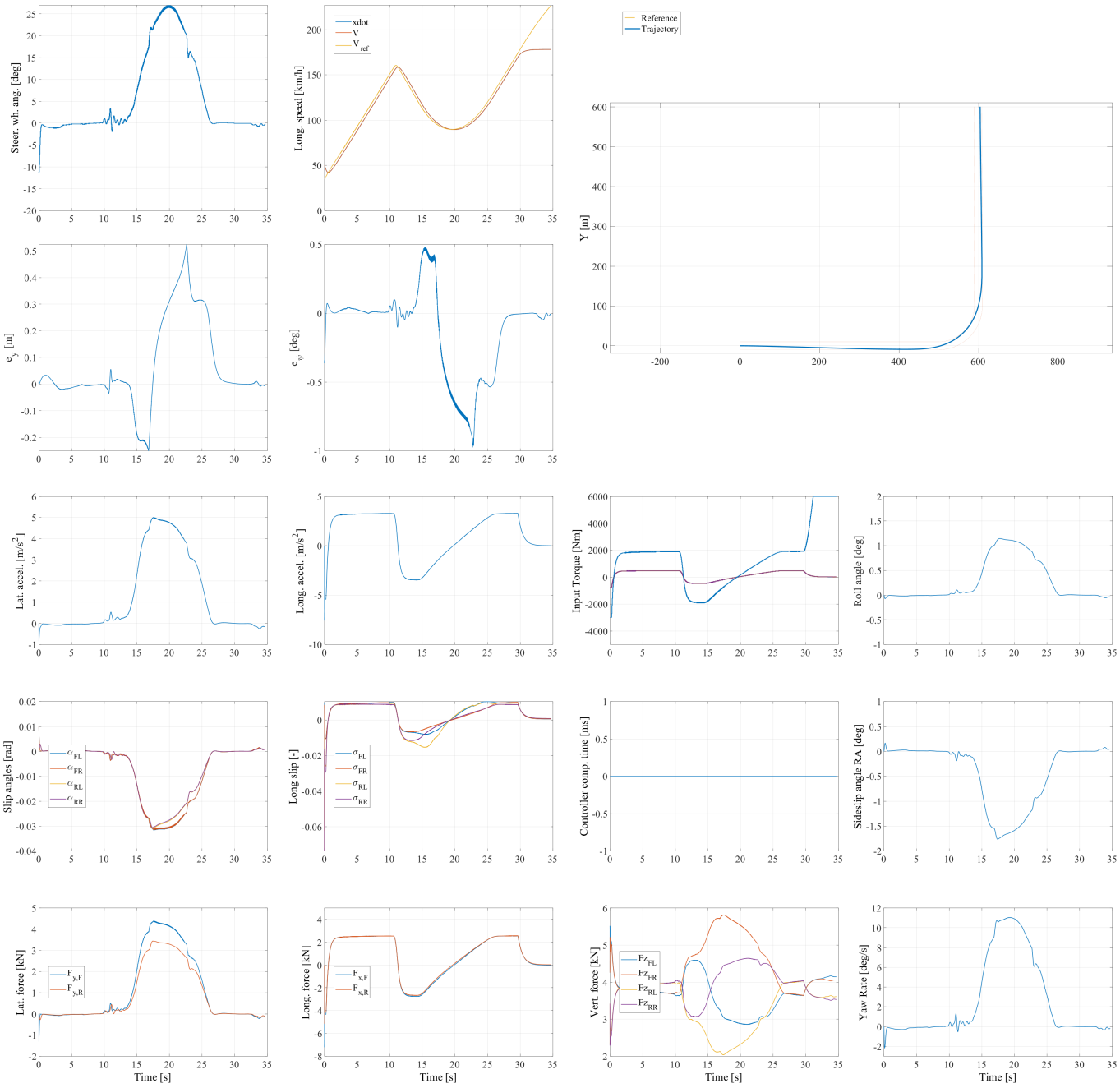

Figure 5.6: Training window of the RL.

Chapter 6

Buffers: memory data storage

In this chapter, the buffer implementation is presented. An initial explanation is presented regarding the idea developed, followed by an explanation of the code developed and its Simulink model. The final objective of the proposed work is to develop the buffer system, and to demonstrate how it works, a training example is presented. In fact, in the current state of the project, the buffer works correctly and the agent is able to start training, however, the tuning of the training and the agent is part of the further development that will be completed in the next stages of the project.

6.1 Description

The idea of the buffer as a memory data storage application arose from the need to have a database that would work online during both writing and reading. For this reason, the creation of .mat files containing data structures in MATLAB was used.

Procedure

During the phase of reading, the buffer is loaded into the workspace at the start of each episode and the values saved from previous laps are loaded into LUTs. These,

based on the $s_{\text{centerline}}$ information of the current lap, interpolate the data saved at the same position but referring to previous laps. In this way, from the current position of the vehicle, it is possible to extrapolate the dynamic information of longitudinal and lateral acceleration, steering angle, vehicle inertial co-ordinates and yaw angle from previous manoeuvres.

Centre-line information

The decision to use $s_{\text{centerline}}$ as input for retrieving saved data was made for several reasons. Firstly, it simplifies the process by requiring only one piece of information, avoiding the need for complex 3D look-up tables (LUTs), simplifying the Simulink model, furthermore makes it possible to read data with information that is comparable with each other, preventing the data read from being discordant with the current vehicle data.

Consider two different driving manoeuvres: an optimised one aiming for the quickest track traversal (e.g., the optimised trajectory) and a highly cautious manoeuvre with a longer duration and less aggressive trajectory (e.g., an initial trial by the agent). To ensure that data saved at a certain point, denoted as k along the track, remains comparable, it's crucial that this data is retrieved at the same spatial position relative to the vehicle.
One might initially think of using the x and y coordinates of the vehicle as inputs, with the dynamic variables from previous laps as output. However, this approach would necessitate the use of computationally intensive 3D LUTs, which would burden the model significantly. Hence, the decision was made to use $s_{\text{centerline}}$ as the reference for comparison.

Making reference to the two-trajectory case hypothesised. Given a point P, defined as a specific point along the curvilinear coordinate after travelling a certain distance k, using the vehicle's instantaneous curvilinear coordinate s would mean that for a slow trajectory (e.g., hugging the centre line), the car would reach point $P1$ after covering k meters. In contrast, for a more aggressive trajectory, the car would reach a different point $P2$ after covering the same k meters, distinct from $P1$.
Assuming that data from these two laps are saved in the buffer and a third trajectory

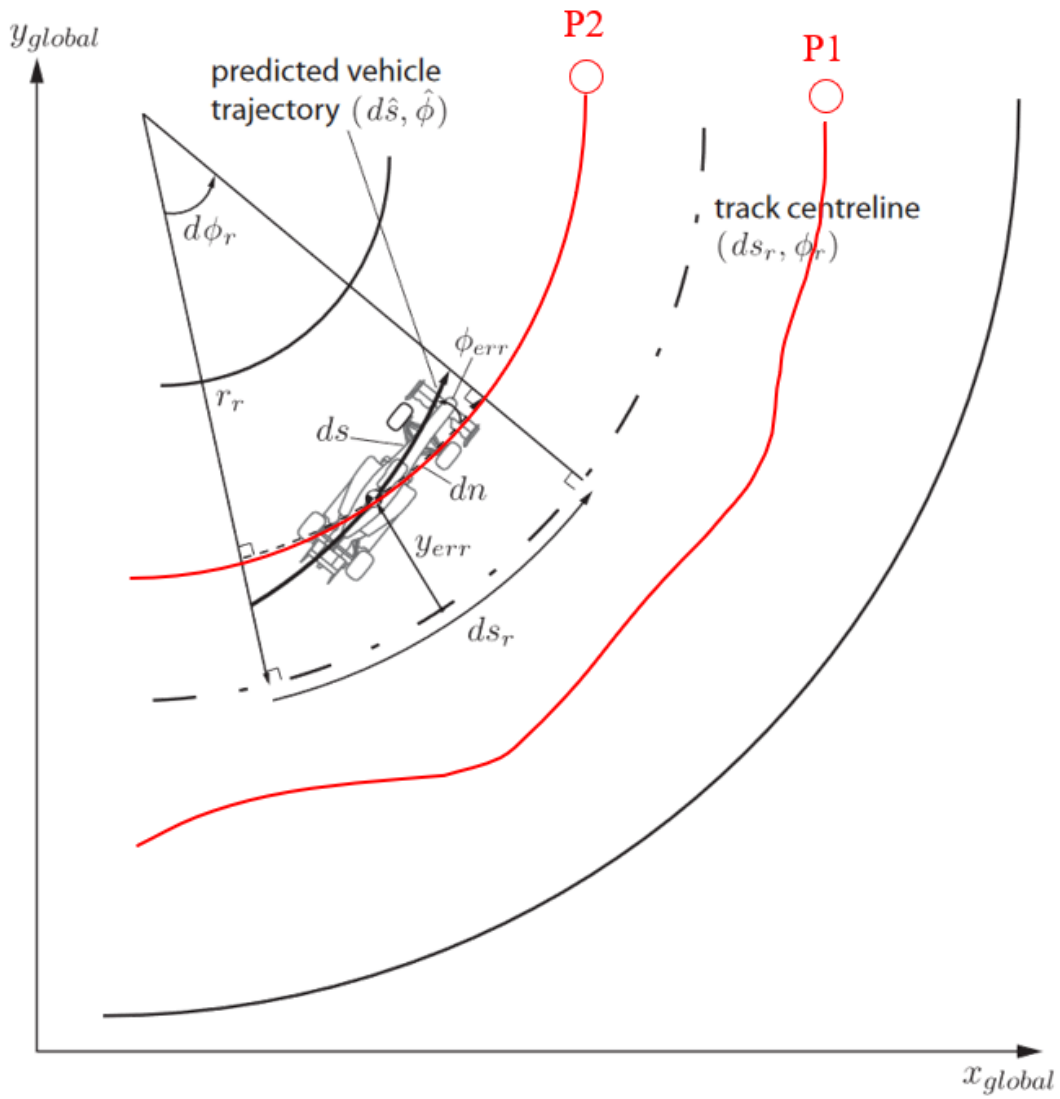

Figure 6.1: Difference among the distance covered by the two trajectories. Path1 will reach $P1$ with a covered distance bigger respect to path2 [23].

is being followed, using s as the read value would result in the agent receiving simultaneous information between $P1$ and $P2$. Since these points are spatially distinct, they might also represent different locations on the track, such as the entry and exit points of a curve. This could lead to mixed observations, such as acceleration (at the curve exit) and deceleration (at the curve entry), occurring simultaneously. Such mixed situations would complicate the training of the agent.

Instead, by utilising the $s_{\text{centerline}}$ information, the algorithm ensures that when the vehicle's projection onto the centre line trajectory covers a distance k, it consults the database at that precise point. Here, all the collected data are directly comparable since they pertain to the same spatial coordinate, occurring under the same driving conditions during the manoeuvre. Moreover, this approach only requires one input for the look-up table, thus simplifying the algorithm.

Moving window

In order to further lighten the algorithm, instead of reading the entire database during the manoeuvre, it was decided to interpret the data readout only for a moving window, i.e. for the values corresponding to the vehicle's current position plus a defined margin of metres forwards and backwards from the current position.

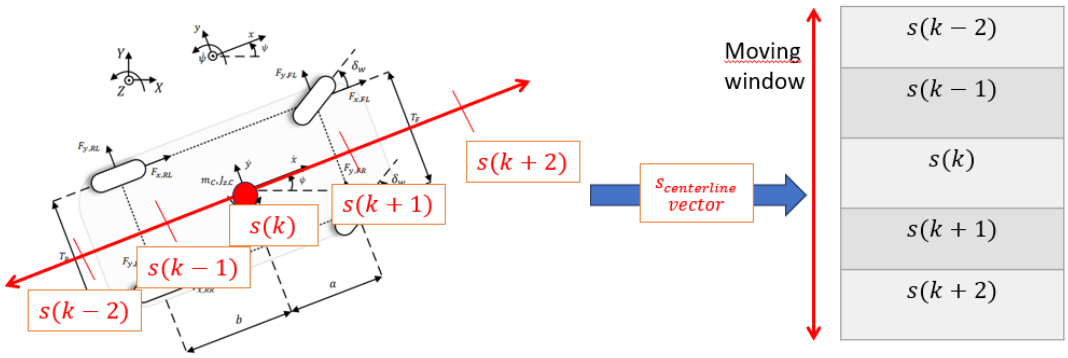

Figure 6.2: Idea used of the moving window.

In order to interpolate the buffer data considering the moving window, it was sufficient to add (in the forward case) or subtract the current position of the vehicle by the defined metres. The size of the window is defined via a parameter that can be set in the MATLAB launch code. In order to have distinct values, the window sizes are defined as vectors, which are concatenated thanks to the `mux`. In this way, if for example a window size of 5 is set, corresponding to the 5 metres in front of and behind the vehicle, a vector of 11 elements will be generated ($[data_{k-5}, ..., data_{k-1} + data_k + data_{k+1}, ..., data_{k+5}]$) and for each of its elements an interpolation will take place associating the corresponding data of the previous laps with each of the

11 elements.

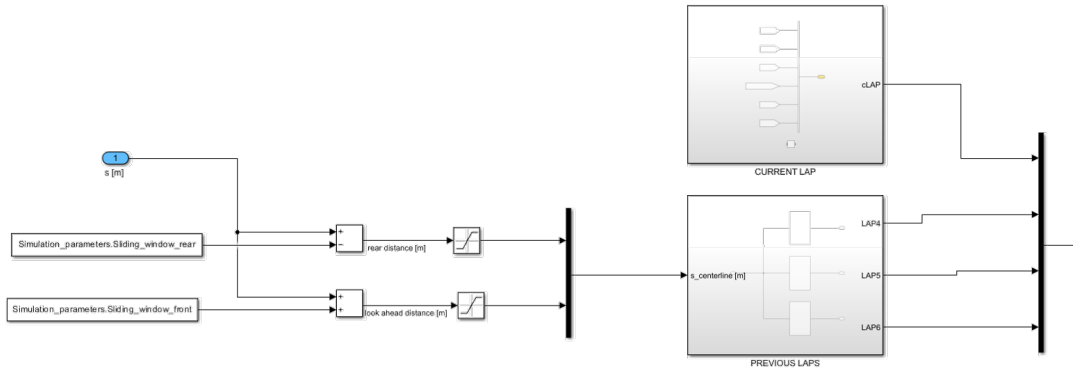

Figure 6.3: Moving window implementation

6.2 Implementation

The buffers were implemented as look-up tables that receive as input the projection of the vehicle's curvilinear co-ordinate along the centre-line trajectory and allow the extraction of the values of the dynamic variables saved in previous laps.

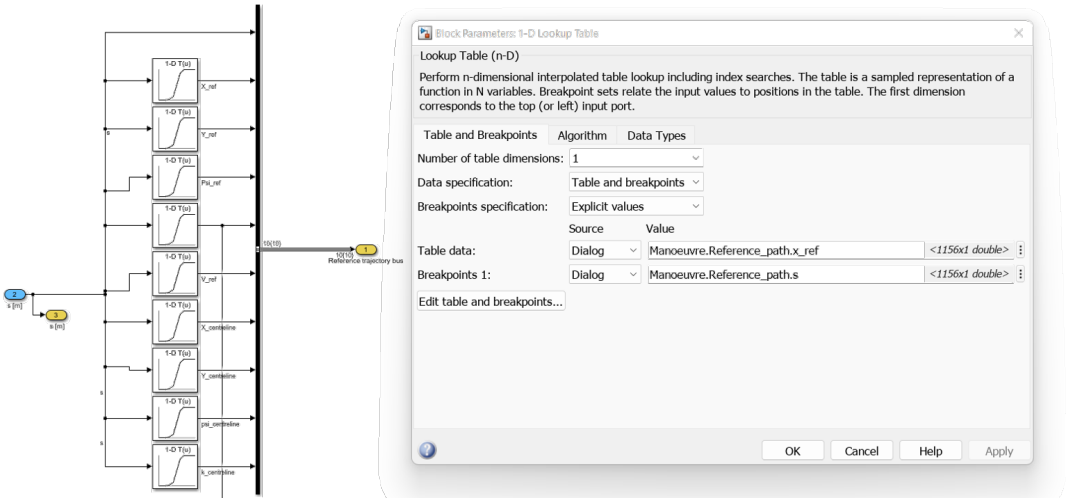

Figure 6.4: Simulink implementation of the look-up tables (buffer data retrieval)

Before any simulation can take place, buffers must be initialised. In fact, it is not

possible to add a variable during training but only to modify an existing one [49]. For the proposed application, it was decided to initialise the buffers to the standard manoeuvre shown in figure 5.6. The results of this simulation are saved in the mat file `Dataset.mat`. This is done in a function (`Get_Buffer_Data.m`) that is launched when the simulation is started (refer to line 75 of the code presented in listing 5.1). The code is presented in the following listing 6.1.

Listing 6.1: Get_Buffer_Data

```
1  function Buffer_Data = Get_Buffer_Data()
2
3  %-------------------------------------------------------------------------
4  % Buffer loading
5  %-------------------------------------------------------------------------
6
7  % In this way a ready-to-go buffer for the 90 degree manoeuvre is
       loaded
8
9  % and fully working.
10
11 %-------------------------------------------------------------------------
12
13      buffer_path = "1_Parameters\07_Buffer\Dataset.mat";
14      load(buffer_path) ;
15
16 %-------------------------------------------------------------------------
17 % Creation of initial signals
18 %-------------------------------------------------------------------------
19
20 % For the first episode there is the need to pre-assign the initial
       signals
21
22 % which are going to be stored into the LAP1 field. This part should
       be
23
24 % executed only if the file "StandardManoeuvreSignals.mat" has been
       lost or
25
26 % corrupted
```

```
27
28 %--------------------------------------------------------------------
29
30     restoreInitialData = false ;
31
32     if restoreInitialData
33
34         InitialSignal.X = X ;
35
36         InitialSignal.Y = Y ;
37
38         InitialSignal.psi = psi_yaw ;
39
40         InitialSignal.delta = delta ;
41
42         InitialSignal.ax = xddot ;
43
44         InitialSignal.ay = yddot ;
45
46         save("1_Parameters\07_Buffer\StandardManoeuvreSignals.mat","
    InitialSignal");
47     end
48
49 %--------------------------------------------------------------------
50 % Loading of initial signals
51 %--------------------------------------------------------------------
52
53 % For the first episode there is the need to pre-assign the initial
    signals
54
55 % which are going to be stored into the LAP1 field. The nomeclature
    follows
56
57 % the same one created and saved in the workspace at the end of the
    episode
58
59 % for coherence
60
61 %--------------------------------------------------------------------
```

```
62
63 load("1_Parameters\07_Buffer\StandardManoeuvreSignals.mat") ;
64
65 X               = InitialSignal.X;
66 Y               = InitialSignal.Y;
67 psi_yaw         = InitialSignal.psi;
68 delta           = InitialSignal.delta;
69 xddot           = InitialSignal.ax;
70 yddot           = InitialSignal.ay;
```

In order to allow the table data of the LUTs to vary with each episode (see the name of the table data in figure 6.4), a localResteFcn was used. This function in MATLAB was idealised for resetting the initial conditions of a given Simulink bloc. However, any variable that is modified internally via MATLAB code can be re-initialised. So in the proposed application, at each episode the data just generated by the simulation is saved in the buffer, which will be read from the moving windows in the next episode. It is therefore essential to have a function that allows this data to be saved at the end of each episode, removing the information from the oldest lap and replacing it with that of the lap just completed. The function which allows this is the localResetFcn whose code is presented in the listings 6.2.

Listing 6.2: localResetFcn

```
1  function in = localResetFcn(in,Buffer_Data,Manoeuvre)
2
3  %-------------------------------------------------------------------
4  %  Buffer uploading
5  %-------------------------------------------------------------------
6
7  % At the end of each episode, the last lap is stored into the LAP1
       field
8
9  % deleting the old LAP3 present. The old LAP2 and LAP1 are stored in
       the
10
11 % successive field. For better clarification makes reference to this
12
13 % scheme:
```

```
14
15 %            current_LAP ---> LAP1
16
17 %            LAP1 (stored in the buffer) ---> LAP2
18
19 %            LAP2 (stored in the buffer) ---> LAP3
20
21 %-------------------------------------------------------------------
22
23 % Scaling the previous LAPS
24
25 Buffer_Data.LAP3 = Buffer_Data.LAP2 ;
26
27 Buffer_Data.LAP2 = Buffer_Data.LAP1 ;
28
29 %-------------------------------------------------------------------
30
31 % Take the signal from the simualtion and resizing them to match size
       for
32
33 % the LUT table. Standard manoeuvre signals are stored in the .mat
      file
34
35 % "StandardManoeuvresSignals"
36
37 load("1_Parameters\07_Buffer\StandardManoeuvreSignals.mat");
38
39 X_currentLap            = changeLength(X,length(Manoeuvre.
       Reference_path.s)) ;
40 Y_currentLap            = changeLength(Y,length(Manoeuvre.
       Reference_path.s)) ;
41 psi_currentLap          = changeLength(psi,length(Manoeuvre.
       Reference_path.s)) ;
42 delta_currentLap        = changeLength(delta,length(Manoeuvre.
       Reference_path.s)) ;
43 ax_currentLap           = changeLength(xddot,length(Manoeuvre.
       Reference_path.s)) ;
44 ay_currentLap           = changeLength(yddot,length(Manoeuvre.
       Reference_path.s)) ;
```

```
45
46 Buffer_Data.LAP1.X            = X_currentLap;
47 Buffer_Data.LAP1.Y            = Y_currentLap;
48 Buffer_Data.LAP1.psi          = psi_currentLap;
49 Buffer_Data.LAP1.delta        = delta_currentLap;
50 Buffer_Data.LAP1.ax           = ax_currentLap;
51 Buffer_Data.LAP1.ay           = ay_currentLap;
52
53 %-------------------------------------------------------------------------
54 %  Buffer saving
55 %-------------------------------------------------------------------------
56
57 % Once the Buffer has been updated, it needs to be saved: TO BE
       CHECKED
58
59 %-------------------------------------------------------------------------
60
61 save("1_Parameters\07_Buffer\Dataset.mat","Buffer_Data");
62
63
64 %-------------------------------------------------------------------------
65 %  Buffer saving
66 %-------------------------------------------------------------------------
67
68 % Put the variable into "in" in order to permit the uploading after
       the
69
70 % episode.
71
72 %-------------------------------------------------------------------------
73
74 in = setVariable(in,'Buffer_Data',Buffer_Data);
75
76 end
```

6.3 Results

The procedure presented thus allows for agent training, the learning outcome of which is shown in the figure 6.5. In this context, the training window of interest is notably the phase during which significant improvements occur, stretching from lap n to lap n+m. The training has been terminated when the improvement in lap time was not high enough to be worth the computational time needed for further episodes. It is noteworthy that the training tendency is a slow convergence towards an asymptote, corresponding the limit beyond which the agent would not be able to improve. This limit is the optimal trajectory of the manoeuvre, which corresponds to the minimum lap time of the simulation posed. Consequently, it is possible to appreciate how the agent used manages to improve and achieve lap times comparable to those of the optimal trajectory.

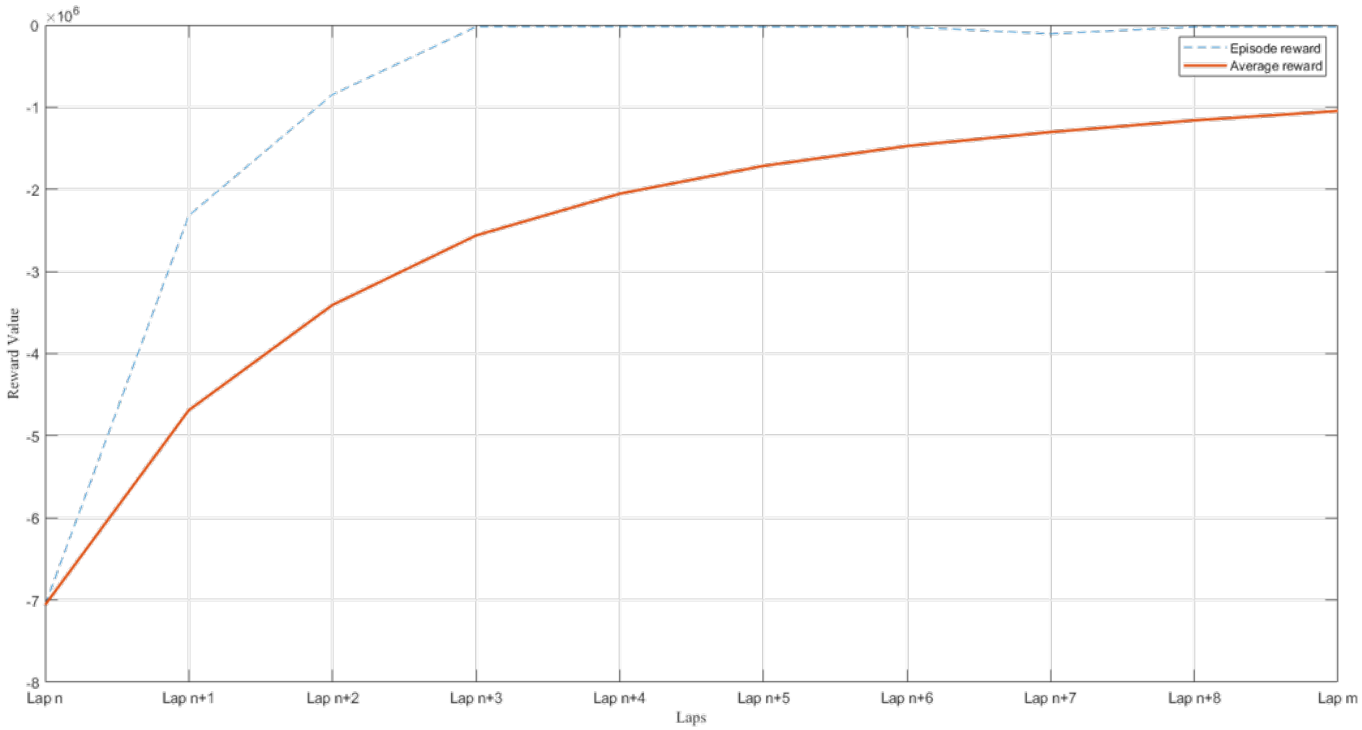

Figure 6.5: Training window of the RL.

The generated trajectories are proposed in figure 6.6. Here one can appreciate how

in the initial stages of training the agent tends not to deviate from the centreline, while the last trained agent shows a tendency towards an optimal trajectory. The final result shows how the agent has the possibility to improve, even if the final trajectory shows slight fluctuations and tends to deviate slightly from the centreline. This shows that the agent still has room for improvement, among which the reward function a term could be inserted to penalise the yaw rate of the vehicle.

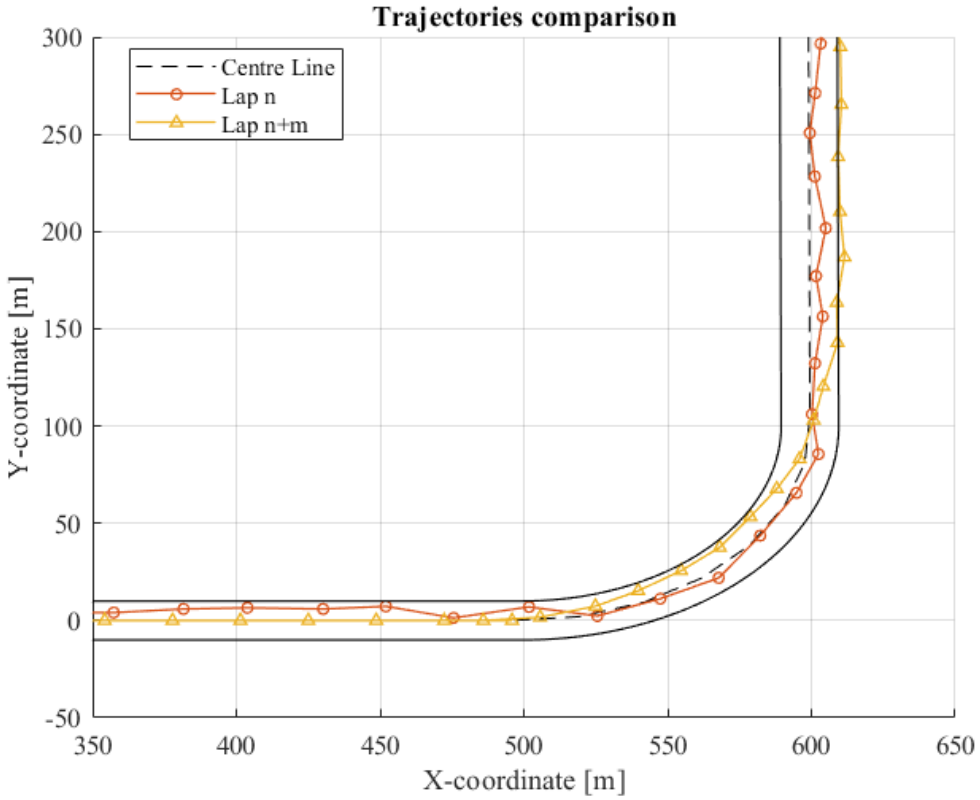

Figure 6.6: Trajectory comparison considering different agents.

Chapter 7

Conclusion

7.1 Novelty points

The work carried out by this thesis thus consists of a state-of-the-art study of the problem of minimum la time simulation. A careful analysis was proposed and summarised in the tables in chapter 2, in which the vehicle modelling, methodology and algorithm used were presented and discussed. Subsequently, a study on the application of Reinforcement Learning as an agent for optimising time as a trajectory generator was carried out. However, since the ultimate goal of the project is to optimise the lap time by taking into account the vehicle's dynamic information, which is updated in real time and also takes into account past information, a lap time simulator which, although realistic, uses a point mass model could be a limitation. For this reason, a move was made towards a more complex model, which allows a more realistic representation of the vehicle, from which the agent's training would benefit in terms of convergence and results. Fundamental to the application of this idea is the need to adopt buffers, which are databases of previous simulations that allow the agent to compare the dynamic variables of the current episode with those of previous laps. Once the buffers had been implemented, and a reward function suitable for the problem posed had been designated, an agent was trained to learn how to generate a delta of curvature which, when added to the curvature of the centre line, allows the path tracking controller to follow the optimised trajectory.

7.2 Further improvements

The work presented achieves satisfactory results in a simple and not too complex manoeuvre. This is due to the computational effort that would require having to simulate and train the agent on longer and more complex circuits. Therefore, the first limitation is the computational availability one has to test more complex trajectories. In addition, one could consider transferring the most complex part of the calculation, such as training the agent, to a computationally less intrusive software than MATLAB-Simulink, however with the complexity of having to interface such a program with the 8 dof vehicle model. Next, an aspect of improvement is the reward function. This needs to be well defined according to the environment in which the agent is located. An analysis of the different weights can be carried out, as the inclusion of other vehicle signals such as lateral and longitudinal acceleration. Still itinerant to the agent, reducing the number of observations certainly represents an achievable improvement, as it would lighten the network, so an analysis on which observations can be avoided can be carried out. Another aspect for discussion could be the implementation of other Reinforcement Learning architectures, such as the latest TD3 algorithm. In conclusion, one of the characteristics of Reinforcement Learning is that since it is a non-deterministic approach, possible new implementations do not necessarily imply better results and the improvement process may be long and not as fruitful as hoped.

Bibliography

[1] Nicola Dal Bianco, Enrico Bertolazzi, Francesco Biral, and Matteo Massaro. «Comparison of direct and indirect methods for minimum lap time optimal control problems». In: *Vehicle System Dynamics* 57.5 (May 4, 2019), pp. 665–696. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2018.1480048. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2018.1480048 (visited on 10/10/2023) (cit. on pp. 1, 13, 16, 19).

[2] William F. Milliken and Douglas L. Milliken. *Race car vehicle dynamics*. Warrendale, PA, U.S.A: SAE International, 1995. 890 pp. ISBN: 978-1-56091-526-3 (cit. on p. 1).

[3] M. Massaro and D. J. N. Limebeer. «Minimum-lap-time optimisation and simulation». In: *Vehicle System Dynamics* 59.7 (July 3, 2021), pp. 1069–1113. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2021.1910718. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2021.1910718 (visited on 10/10/2023) (cit. on pp. 1, 2, 4–7, 9, 14, 17, 22).

[4] D. Metz and D. Williams. «Near time-optimal control of racing vehicles». In: *Automatica* 25.6 (Nov. 1989), pp. 841–857. ISSN: 00051098. DOI: 10.1016/0005-1098(89)90052-6. URL: https://linkinghub.elsevier.com/retrieve/pii/0005109889900526 (visited on 07/31/2023) (cit. on pp. 2, 6, 7, 9, 10, 14, 15).

[5] Xiaohui Hou, Junzhi Zhang, Chengkun He, Yuan Ji, Junfeng Zhang, and Jinheng Han. «Autonomous driving at the handling limit using residual reinforcement learning». In: *Advanced Engineering Informatics* 54 (Oct. 2022), p. 101754. ISSN: 14740346. DOI: 10.1016/j.aei.2022.101754. URL: https:

//linkinghub.elsevier.com/retrieve/pii/S1474034622002129 (visited on 10/10/2023) (cit. on pp. 2, 6, 24).

[6] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Durr. «Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning». In: *IEEE Robotics and Automation Letters* 6.3 (July 2021), pp. 4257–4264. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3064284. URL: https://ieeexplore.ieee.org/document/9372847/ (visited on 10/10/2023) (cit. on pp. 2, 22, 23).

[7] Achin Jain and Manfred Morari. *Computing the racing line using Bayesian optimization*. Feb. 11, 2020. arXiv: 2002.04794[cs]. URL: http://arxiv.org/abs/2002.04794 (visited on 10/10/2023) (cit. on pp. 5, 21).

[8] Mathworks - ETH. *Lap Time Simulation; Essential Part of Concept Development*. 2014. URL: https://it.mathworks.com/videos/matlab-and-simulink-racing-lounge-lap-time-simulation-essential-part-of-concept-development-98733.html (cit. on p. 6).

[9] SAE international. *Formula SAE regulation - 2022*. 2022. URL: https://fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=25e8885c-7397-4b2d-93b3-fc404960bab1 (cit. on p. 6).

[10] M. Veneri and M. Massaro. «A free-trajectory quasi-steady-state optimal-control method for minimum lap-time of race vehicles». In: *Vehicle System Dynamics* 58.6 (June 2, 2020), pp. 933–954. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2019.1608364. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1608364 (visited on 10/10/2023) (cit. on p. 8).

[11] Marco Gadola, David Vetturi, Danilo Cambiaghi, and Luca Manzo. «A Tool for Lap Time Simulation». In: Motorsports Engineering Conference & Exposition. Dec. 1, 1996, p. 962529. DOI: 10.4271/962529. URL: https://www.sae.org/content/962529/ (visited on 10/10/2023) (cit. on pp. 10, 11, 21).

[12] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. «Minimum curvature trajectory planning and control for an autonomous race car». In: *Vehicle System Dynamics* 58.10 (Oct. 2, 2020), pp. 1497–1527. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2019.1631455. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1631455 (visited on 07/31/2023) (cit. on pp. 11, 12, 22, 53).

[13] Miguel Morales. *Grokking Deep Reinforcement Learning*. Shelter Island, New York: Manning Publications, 2020. 472 pp. ISBN: 978-1-61729-545-4 (cit. on pp. 14, 26, 28, 29, 35, 53).

[14] Mattia Zanchetta. «Autonomous Driving and Stability Control of Over-Actuated Vehicles at the Limits of Handling». PhD thesis. University of Surrey, Dec. 2019 (cit. on pp. 16, 59, 61).

[15] Giacomo Perantoni and David J.N. Limebeer. «Optimal control for a Formula One car with variable parameters». In: *Vehicle System Dynamics* 52.5 (May 4, 2014), pp. 653–678. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2014.889315. URL: http://www.tandfonline.com/doi/abs/10.1080/00423114.2014.889315 (visited on 10/10/2023) (cit. on pp. 19, 21).

[16] Roberto Lot and Nicola Dal Bianco. «Lap time optimisation of a racing go-kart». In: *Vehicle System Dynamics* 54.2 (Feb. 2016), pp. 210–230. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2015.1125514. URL: http://www.tandfonline.com/doi/full/10.1080/00423114.2015.1125514 (visited on 10/10/2023) (cit. on pp. 19, 21).

[17] L. Leonelli and D. J. N. Limebeer. «Optimal control of a road racing motorcycle on a three-dimensional closed track». In: *Vehicle System Dynamics* 58.8 (Aug. 2, 2020), pp. 1285–1309. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2019.1617886. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1617886 (visited on 10/10/2023) (cit. on pp. 19, 22).

[18] J.P.M. Hendrikx, T.J.J. Meijlink, and R.F.C. Kriens. «Application of Optimal Control Theory to Inverse Simulation of Car Handling». In: *Vehicle System Dynamics* 26.6 (Dec. 1996), pp. 449–461. ISSN: 0042-3114, 1744-5159. DOI:

10.1080/00423119608969319. URL: http://www.tandfonline.com/doi/abs/10.1080/00423119608969319 (visited on 09/17/2023) (cit. on p. 21).

[19] V. Cossalter, M. Da Lio, R. Lot, and L. Fabbri. «A General Method for the Evaluation of Vehicle Manoeuvrability with Special Emphasis on Motorcycles». In: *Vehicle System Dynamics* 31.2 (Feb. 1, 1999), pp. 113–135. ISSN: 0042-3114. DOI: 10.1076/vesd.31.2.113.2094. URL: http://www.tandfonline.com/doi/abs/10.1076/vesd.31.2.113.2094 (visited on 09/17/2023) (cit. on p. 21).

[20] Casanova. «On minimum time vehicle manoeuvring: the theoretical optimal lap». PhD thesis. Cranfield University, 2000. URL: http://hdl.handle.net/1826/1091 (cit. on p. 21).

[21] D L Brayshaw and M F Harrison. «A quasi steady state approach to race car lap simulation in order to understand the effects of racing line and centre of gravity location». In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 219.6 (June 1, 2005), pp. 725–739. ISSN: 0954-4070, 2041-2991. DOI: 10.1243/095440705X11211. URL: http://journals.sagepub.com/doi/10.1243/095440705X11211 (visited on 10/10/2023) (cit. on p. 21).

[22] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. «Computing minimum lap-time trajectories for a single-track car with load transfer». In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. 2012 IEEE 51st Annual Conference on Decision and Control (CDC). Maui, HI, USA: IEEE, Dec. 2012, pp. 6321–6326. DOI: 10.1109/CDC.2012.6426265. URL: http://ieeexplore.ieee.org/document/6426265/ (visited on 10/10/2023) (cit. on pp. 21, 24).

[23] Julian P. Timings and David J. Cole. «Vehicle trajectory linearisation to enable efficient optimisation of the constant speed racing line». In: *Vehicle System Dynamics* 50.6 (June 2012), pp. 883–901. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2012.671946. URL: http://www.tandfonline.com/doi/abs/10.1080/00423114.2012.671946 (visited on 07/31/2023) (cit. on pp. 21, 24, 69).

[24] Paul A. Theodosis and J. Christian Gerdes. «Nonlinear Optimization of a Racing Line for an Autonomous Racecar Using Professional Driving Techniques». In: *Volume 1: Adaptive Control; Advanced Vehicle Propulsion Systems; Aerospace Systems; Autonomous Systems; Battery Modeling; Biochemical Systems; Control Over Networks; Control Systems Design; Cooperativ*. ASME 2012 5th Annual Dynamic Systems and Control Conference joint with the JSME 2012 11th Motion and Vibration Conference. Fort Lauderdale, Florida, USA: ASME, Oct. 17, 2012, pp. 235–241. ISBN: 978-0-7918-4529-5. DOI: 10.1115/DSCC2012-MOVIC2012-8620. URL: http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?doi=10.1115/DSCC2012-MOVIC2012-8620 (visited on 10/10/2023) (cit. on p. 21).

[25] Maximilian Brunner, Ugo Rosolia, Jon Gonzales, and Francesco Borrelli. «Repetitive learning model predictive control: An autonomous racing example». In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017 IEEE 56th Annual Conference on Decision and Control (CDC). Melbourne, Australia: IEEE, Dec. 2017, pp. 2545–2550. ISBN: 978-1-5090-2873-3. DOI: 10.1109/CDC.2017.8264027. URL: http://ieeexplore.ieee.org/document/8264027/ (visited on 10/10/2023) (cit. on p. 21).

[26] Danio Caporale et al. «A Planning and Control System for Self-Driving Racing Vehicles». In: *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*. 2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI). Palermo: IEEE, Sept. 2018, pp. 1–6. ISBN: 978-1-5386-6282-3. DOI: 10.1109/RTSI.2018.8548444. URL: https://ieeexplore.ieee.org/document/8548444/ (visited on 10/10/2023) (cit. on p. 21).

[27] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N. Zeilinger. «Learning-Based Model Predictive Control for Autonomous Racing». In: *IEEE Robotics and Automation Letters* 4.4 (Oct. 2019), pp. 3363–3370. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2019.2926677. URL: https://ieeexplore.ieee.org/document/8754713/ (visited on 10/10/2023) (cit. on p. 21).

[28] Alexandra Tătulea-Codrean, Tommaso Mariani, and Sebastian Engell. «Design and Simulation of a Machine-learning and Model Predictive Control Approach to Autonomous Race Driving for the F1/10 Platform». In: *IFAC-PapersOnLine* 53.2 (2020), pp. 6031–6036. ISSN: 24058963. DOI: 10.1016/j.ifacol.2020.12.1669. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405896320322722 (visited on 10/10/2023) (cit. on p. 21).

[29] Ugo Rosolia and Francesco Borrelli. «Learning How to Autonomously Race a Car: A Predictive Control Approach». In: *IEEE Transactions on Control Systems Technology* 28.6 (Nov. 2020), pp. 2713–2719. ISSN: 1063-6536, 1558-0865, 2374-0159. DOI: 10.1109/TCST.2019.2948135. URL: https://ieeexplore.ieee.org/document/8896988/ (visited on 10/10/2023) (cit. on p. 22).

[30] Emilio Capo and Daniele Loiacono. «Short-Term Trajectory Planning in TORCS using Deep Reinforcement Learning». In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020 IEEE Symposium Series on Computational Intelligence (SSCI). Canberra, ACT, Australia: IEEE, Dec. 1, 2020, pp. 2327–2334. ISBN: 978-1-72812-547-3. DOI: 10.1109/SSCI47803.2020.9308138. URL: https://ieeexplore.ieee.org/document/9308138/ (visited on 10/10/2023) (cit. on p. 23).

[31] Sam Garlick and Andrew Bradley. «Real-time optimal trajectory planning for autonomous vehicles and lap time simulation using machine learning». In: *Vehicle System Dynamics* (2021). DOI: 10.1080/00423114.2021.2011929 (cit. on p. 23).

[32] Fabian Christ, Alexander Wischnewski, Alexander Heilmeier, and Boris Lohmann. «Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients». In: *Vehicle System Dynamics* 59.4 (Apr. 3, 2021), pp. 588–612. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2019.1704804. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1704804 (visited on 10/10/2023) (cit. on p. 23).

[33] Eugenio Chisari, Alexander Liniger, Alisa Rupenyan, Luc Van Gool, and John Lygeros. *Learning from Simulation, Racing in Reality*. May 7, 2021. arXiv: 2011.13332[cs,eess]. URL: http://arxiv.org/abs/2011.13332 (visited on 10/10/2023) (cit. on p. 23).

[34] Peide Cai, Hengli Wang, Huaiyang Huang, Yuxuan Liu, and Ming Liu. *Vision-Based Autonomous Car Racing Using Deep Imitative Reinforcement Learning*. July 17, 2021. arXiv: 2107.08325[cs,eess]. URL: http://arxiv.org/abs/2107.08325 (visited on 10/10/2023) (cit. on p. 23).

[35] Olaf Borsboom, Chyannie Amarillio Fahdzyana, Theo Hofman, and Mauro Salazar. «A Convex Optimization Framework for Minimum Lap Time Design and Control of Electric Race Cars». In: *IEEE Transactions on Vehicular Technology* 70.9 (Sept. 2021), pp. 8478–8489. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2021.3093164. URL: https://ieeexplore.ieee.org/document/9468407/ (visited on 10/10/2023) (cit. on p. 23).

[36] Jaroslav Klapalek, Antonin Novak, Michal Sojka, and Zdenek Hanzalek. «Car Racing Line Optimization with Genetic Algorithm using Approximate Homeomorphism». In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Prague, Czech Republic: IEEE, Sept. 27, 2021, pp. 601–607. ISBN: 978-1-66541-714-3. DOI: 10.1109/IROS51168.2021.9636503. URL: https://ieeexplore.ieee.org/document/9636503/ (visited on 10/10/2023) (cit. on p. 23).

[37] Axel Brunnbauer, Luigi Berducci, Andreas Brandstätter, Mathias Lechner, Ramin Hasani, Daniela Rus, and Radu Grosu. *Latent Imagination Facilitates Zero-Shot Transfer in Autonomous Racing*. Feb. 28, 2022. arXiv: 2103.04909[cs]. URL: http://arxiv.org/abs/2103.04909 (visited on 10/10/2023) (cit. on p. 23).

[38] Adrian Remonda, Sarah Krebs, Eduardo Veas, Granit Luzhnica, and Roman Kern. *Formula RL: Deep Reinforcement Learning for Autonomous Racing using Telemetry Data*. June 13, 2022. arXiv: 2104.11106[cs]. URL: http://arxiv.org/abs/2104.11106 (visited on 10/10/2023) (cit. on p. 23).

[39] Stan Broere, Jorn Van Kampen, and Mauro Salazar. «Minimum-lap-time Control Strategies for All-wheel Drive Electric Race Cars via Convex Optimization». In: *2022 European Control Conference (ECC)*. 2022 European Control Conference (ECC). London, United Kingdom: IEEE, July 12, 2022, pp. 1204–1211. ISBN: 978-3-907144-07-7. DOI: 10.23919/ECC55457.2022.9838115.

URL: https://ieeexplore.ieee.org/document/9838115/ (visited on 10/10/2023) (cit. on p. 23).

[40] Pieter De Buck and Joaquim R. R. A Martins. «Minimum lap time trajectory optimisation of performance vehicles with four-wheel drive and active aerodynamic control». In: *Vehicle System Dynamics* (July 19, 2022), pp. 1–17. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2022.2101930. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2022.2101930 (visited on 10/10/2023) (cit. on p. 24).

[41] K. Tucker, R. Gover, R. N. Jazar, and H. Marzbani. «Feasible trajectory planning for minimum time manoeuvring». In: *Vehicle System Dynamics* (Jan. 9, 2023), pp. 1–32. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2022.2164314. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2022.2164314 (visited on 10/10/2023) (cit. on p. 24).

[42] Mattia Piccinini, Matteo Larcher, Edoardo Pagot, Davide Piscini, Leone Pasquato, and Francesco Biral. «A predictive neural hierarchical framework for on-line time-optimal motion planning and control of black-box vehicle models». In: *Vehicle System Dynamics* 61.1 (Jan. 2, 2023), pp. 83–110. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2022.2035776. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2022.2035776 (visited on 10/10/2023) (cit. on p. 24).

[43] J. Biniewicz and M. Pyrz. «A quasi-steady-state minimum lap time simulation of race motorcycles using experimental data». In: *Vehicle System Dynamics* (Jan. 31, 2023), pp. 1–23. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2023.2170256. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2023.2170256 (visited on 10/10/2023) (cit. on p. 24).

[44] Angelo Borneo, Luca Zerbato, Federico Miretti, Antonio Tota, Enrico Galvagno, and Daniela Anna Misul. «Platooning Cooperative Adaptive Cruise Control for Dynamic Performance and Energy Saving: A Comparative Study of Linear Quadratic and Reinforcement Learning-Based Controllers». In: *Applied Sciences* 13.18 (Sept. 19, 2023), p. 10459. ISSN: 2076-3417. DOI: 10.3390/app131810459. URL: https://www.mdpi.com/2076-3417/13/18/10459 (visited on 09/25/2023) (cit. on p. 26).

[45] Ian Goodfellow, Bengio Yoshua, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (cit. on p. 27).

[46] Ben Lau. *Using Keras and Deep Deterministic Policy Gradient to play TORCS*. 2016. URL: https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html (cit. on p. 29).

[47] Mathworks. *Deep Deterministic Policy Gradient (DDPG) Agents*. 2023. URL: https://it.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html (cit. on p. 31).

[48] OpenAI. *Deep Deterministic Policy Gradient*. 2022. URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html (cit. on pp. 31, 32).

[49] Mathworks. *fast-restart*. 2023. URL: https://it.mathworks.com/help/sldo/ref/sdo.simulationtest.fastrestart.html?searchHighlight=fast%20restart&s_tid=srchtitle_support_results_2_fast%20restart (cit. on pp. 48, 72).

[50] Mathworks. *workspace*. 2023. URL: https://it.mathworks.com/help/matlab/learn_matlab/workspace.html?searchHighlight=workspace&s_tid=srchtitle_support_results_1_workspace (cit. on p. 48).

[51] Richard Ernest Bellman. *Dynamic Programming*. Courier Dover Publications, 2003. ISBN: 978-0-486-42809-3 (cit. on p. 55).