Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: FSAE aerodynamics and lap time simulator

TITULACIÓ: Doble titulació en Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació i Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Rubén Batanero Ramírez

DIRECTORS: Adeline de Villardi de Montlaur i Miguel Valero Garcia

DATA: 5 de setembre del 2022

Títol: Aerodinàmica d'un FSAE i simulador de temps de volta

Autor: Rubén Batanero Ramírez

Directors: Adeline de Villardi de Montlaur i Miguel Valero Garcia

Data: 5 de setembre del 2022

Resum

FSAE o Formula Student és una competició global entre estudiants universitaris. En aquesta, els estudiants han de crear un cotxe de carreres des de zero, dissenyant la part elèctrica, la part de combustió, dinàmica i aerodinàmica. A més, han d'aconseguir finançar el projecte per poder construir-lo i anar a competir.

L'equip de Formula Student de la Universitat Politècnica de Vilanova i la Geltrú, el Vilanova Formula Team (VFT), ha estat participant en la competició des de 2015, i aquest any estan disposats a fabricar una bona aerodinàmica. Però, el seu departament d'aerodinàmica és bastant nou i és necessari crear una base.

L'aerodinàmica és un dels elements clau en un cotxe de competició. Aquesta permet que aquest arribi a velocitats màximes superiors i que pugui agafar les corbes a més velocitat. Però, aquesta també és un dels aspectes més complexos a l'hora de dissenyar un cotxe, i ve acompanyada de moltes hores de simulacions.

En aquest TFG s'ha dissenyat l'aerodinàmica del cotxe que utilitzarà l'equip de Vilanova en la temporada 2022. El disseny s'ha realitzat amb una base de coneixements aerodinàmics i s'ha adaptat mitjançant simulacions Computational Fluid Dynamics (CFD) en Solidworks. El disseny s'ha fet seguint la normativa de la competició i una constant comunicació amb l'equip per assegurar que els dissenys siguin compatibles amb la resta de departaments. A més, encara que l'estudi de fabricació no estigui explicat en aquest treball, s'ha tingut en compte en tot moment que les peces han de poder ser fabricades.

Finalment, s'ha dissenyat una eina Deep Learning, a partir de xarxes neuronals i algoritmes genètics, per simular els tests a circuit i obtenir els temps de volta que pot obtenir el cotxe amb l'aerodinàmica dissenyada. L'objectiu d'aquesta eina és facilitar als equips que no tenen els recursos necessaris per fabricar múltiples aerodinàmiques i test a pista, una alternativa per poder escollir quina aerodinàmica fabricar.

Title: FSAE aerodynamics and lap time simulator

Author: Rubén Batanero Ramírez

Directors: Adeline de Villardi de Montlaur i Miguel Valero Garcia

Date: September 5, 2022

Overview

FSAE or Formula Student is a worldwide competition between university students. The objective is to create a racing car from scratch, designing its electric circuits, combustion, dynamics and aerodynamics. Moreover, the students need to find the funding to build the car and compete.

The Formula Student Team from the Polytechnic University of Vilanova i la Geltrú, called Vilanova Formula Team (VFT), has been participating in the competition since 2015, and this year they want to build a good aerodynamic for their car. But, the aerodynamic department of the team is still very new and a solid basis is needed.

Aerodynamics in a racing car is one of the key elements during the competition. It helps the car reach higher maximum speeds and take corners at higher velocities. But, it is also one of the most complex aspects of designing a competition car and it requires a significant amount of simulation time.

In this TFG, the aerodynamic devices for the VFT 2022 season car have haven designed from a base of aerodynamic knowledge and optimized using Computational Fluid Dynamics (CFD) simulations in Solidworks. It has been ensured that all the designs follow the competition rules and compare compatible with the other team departments. To do so, there has been constant communication with the team to verify the designs. Moreover, even though this document does not explain the fabrication process, it has been taken into account that one requirement is to create pieces that can be manufactured by the team.

Finally, a Deep Learning tool has been created, using Neural Networks and Genetic Algorithms, to test the designed aerodynamics in a simulated circuit and obtain the fastest Lap time in a determined track. The objective of this tool is to help low-budget teams to test the aerodynamic performance on a race track without needing to build anything.

To my teammates in the
Vilanova Formula team
for introducing me
to this world.

Thanks to my father
for lending me a
computer to run
more than 800
hours of simulations.

And thanks to the
directors of this
project that
accepted my idea
and helped me
responding all
my emails.

ÍNDEX

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

- **Airfoil:** 2d structure with curved surfaces designed to generate lift from the air motion with a favorable lift to drag ratio. A wing is the result of making an airfoil 3d.

- **Algorithm:** set of instructions given to a computer that calculates an answer to a problem.

- **Apex**: in a racing track is the point of a corner at the inside portion of it that a vehicle passes closest to.

- **CFD or Computational fluid dynamics:** computational software that applies physics and mathematics to visualize how a fluid flows.

- **Cockpit:** closed space of a car where the pilot sits.

- **CPU time:** time the computer needs to run a simulation.

- **Deep learning:** use of algorithms similar to the way the human brain operates.

- **Formula Student / FSAE:** worldwide competition in which university students compete to design and build a formula-style single-seated competition car.

- **FSS or Formula Student Spain**: competition part of the Formula Student competition hosted in Spain.

- **Ground Effect:** positive influence on the aerodynamics when a device is close to the ground.

- **HUD or Heads-Up Display:** is a type of user interface commonly used in videogames where the information is visually showed to the user while playing.

- **Keep-out-zones:** 3D volumes in which it is not allowed to build anything.

- **Mach number:** represents the ratio of flow velocity to the local speed of sound.

- **MSHD or Motor Sports High Downforce:** high downforce airfoil designed by PAKKAM, SRIRAM SARANATHY (see [1]).

- **Nova 7:** VFT official competition car for the 2021-2022 season.

- **Open-wheeled vehicle:** car with the wheels outside the car's main body.

- **QoL:** features that make the use of a program easier for the user.

- **Racing line:** optimal path around a race course.

- **Rake:** slope created by raising the rear part of a car or by lowering the front area.

- **VFT or Vilanova Formula Team:** Formula student team from the Polytechnic School of Engineering of Vilanova i la Geltrú.

1. INTRODUCTION

This project covers the aerodynamic design of a formula-style car for a formula student competition, from the initial ideas to the development of a Deep Learning testing tool to compare different aerodynamic designs.

This chapter explains the origins of the problem this project tries to solve and the motivation to do it. The contents of the following chapters and programs used for the project are also briefly described.

1.1. Motivation and statement of the problem

Being part of a Formula Student team since 2019 helped to learn and share a singular engineering experience. One realizes the complexity of a competition's car. Since the team has never had a proper aerodynamic study, this work will try to provide a full car's aerodynamics design in the hope of setting some guidelines for future team members.

The idea of a program, capable of generating the perfect aerodynamic device, by the aggregation of cubes, and assist the aerodynamic design was contemplated. But this idea seemed rather complex, and while studying the problem, an idea appeared related to deep learning. After digging into deep learning algorithms, it was decided to create a lap time simulator instead of a device generator.

A lap time simulator is no less complex than the original idea but it is more practical. One of the problems when designing the aerodynamics of a car is to know how useful each design is. Using aerodynamic simulations, one could obtain the aerodynamic forces of each configuration, but what is the best configuration for each circuit?

Some aerodynamic devices are better for straight lines and some are better for turns. Being a circuit a combination of both, a mix between straight and turn efficiency should be found. The lap time simulator uses Neural networks and Genetic algorithms to find the best lap time a certain car can achieve on a determined track.

1.2. Background

It is important to know the different spheres this project goes into to fulfil its goal. It involves different fields of knowledge such as car dynamics, car aerodynamics and deep learning algorithms.

As it will be shown later, car physics will be applied into the lap time simulator from scratch. This comprehends the study of how a car moves, what happens when it turns, how it accelerates and brakes, how aerodynamics affects the car

movement and when it slides or skids. The understanding of all these concepts allowed the creation of a simplified version of a realistic car simulator that uses aerodynamics.

The need to make the lap time simulator faster and more efficient implied the study of Genetic Algorithms and Neural Networks, their variations and optimization methods.

Finally, most of the project is centered on aerodynamics and therefore most of the background knowledge is focused on this subject. The aerodynamic study is based on Computational Fluid Dynamics (CFD) simulations which implies mesh refining, proper boundary/initial conditions selection and analyzing the simulation results.

1.3. Document structure

This section of the introduction chapter describes what the reader can expect from each chapter of the document.

Chapter 2 (The Competition): this second chapter introduces the team for which the car designed in this project is for, the events the car will face and the basic rules for the design of the car aerodynamic.

Chapter 3 (Aerodynamics Introduction): the third chapter introduces the basic knowledge of aerodynamics that will be needed to understand the project correctly. It explains classic aerodynamics concepts such as Lift or Drag and ends with an explanation of how aerodynamics influence a racing car.

Chapter 4 (Algorithms): the fourth chapter is about the algorithms used to create the lap time simulator. As the previous chapter, this is a theoretical chapter that is needed to understand the development of the project. The two concepts explained are the Neural Network and the Genetic Algorithm.

Chapter 5 (Planning): after all the theory, this fifth chapter introduces the detailed objectives the project is trying to achieve and a work plan of how the tasks were distributed.

Chapter 6 (CFD Methodology): the sixth chapter starts with an explanation of the knowledge necessary to understand the CFD simulations and ends with the results of the simulation configuration. This chapter allows to do the CFD simulations used in the following chapters.

Chapter 7 (FSAE Aerodynamics Design): in the seventh chapter, each aerodynamic device is developed from scratch and tested to create the car. This is the core development of the project and the results of this chapter will be used for the next ones.

Chapter 8 (Aerodynamic Improvement): the eighth chapter explains how the aerodynamics of the previous chapter can be further improved. Most of the ideas in this project could be used for future development in the next car generations.

Chapter 9 (Self-Driving Car): this ninth chapter is the base for the lap time simulator. In this chapter, the program is developed to the point that it can learn to drive a car by applying the algorithms explained in chapter 4.

Chapter 10 (Experimentation): the tenth chapter uses the program developed in the previous chapter and improves it to the point it can drive like a racing pilot and the computing time is reduced. Moreover, the program is improved with Quality of Life (QoL) features to be more user-friendly.

Chapter 11 (Final Results): here, in the eleventh chapter, the aerodynamic results from chapters 8 and 9 are introduced to the lap time simulator to compare them.

Chapter 12 (Final Thoughts): the twelfth chapter is the last chapter of the document and contains the conclusion.

1.4. Programs

During the project, a variety of programs are used to assist in its development. In this section, these programs are briefly described to help the reader understand the purpose of each one inside the project.

1.4.1. Solidworks

Solidworks is a 3D computer-aided design (CAD) software developed by Dassault Systemes. It has a free version for Students and is one of the most used CAD programs in engineering worldwide. This program will be used to design the aerodynamic devices.

The program includes a Flow Simulation Add-On, a Computational Fluid Dynamics (CFD) solution embedded within. After research on the internet, it was concluded that this CFD simulator offers reliable results. Because of it, and that it is included in the Students pack, it is the CFD simulator used in this project.

Note that other CFD programs were considered but a free version is rather difficult to obtain and Solidworks Flow Simulation is reliable enough for the purpose of this project.

All the products can be obtained from the official websites (see [2] and [3]).

1.4.2. Unity

Unity is a videogame engine created by Unity Technologies compatible with the programming languages C# and C++. It can be downloaded for free from the official website (see [4]).

Unity will be used for the lap time simulator. The simulator works in a 3D environment and Unity is compatible. But this program is selected over other alternatives because of the facilities C# will bring when coding the algorithms.

1.4.3. Blender

Blender is an open-source program specialized in 3D modelling, animation and rendering. It is one of the most used programs for 3D art. And it can be downloaded for free on the official website (see [5]) or on Steam.

Blender will be used to create the 3D objects used by the lap time simulator. It is possible to export the 3D designs from Solidworks to Unity but the conversion makes the object heavy to render and therefore it slows the simulator. Blender is preferable because it is compatible with Unity and this will allow creating low-poly ngobjects lighter to render.

Note that the lap time simulator does not use the shape of the aerodynamics when running. The aerodynamic results will be extracted from the Solidworks flow simulations to the Unity engine.

In a nutshell, Solidworks is used to create aerodynamic devices and do CFD simulations. The results then are passed into Unity where the Lap Time simulator is made. And Blender is used to create the 3D objects used as the Lap Time simulator environment such as the race track **(see Fig. 1.1).**

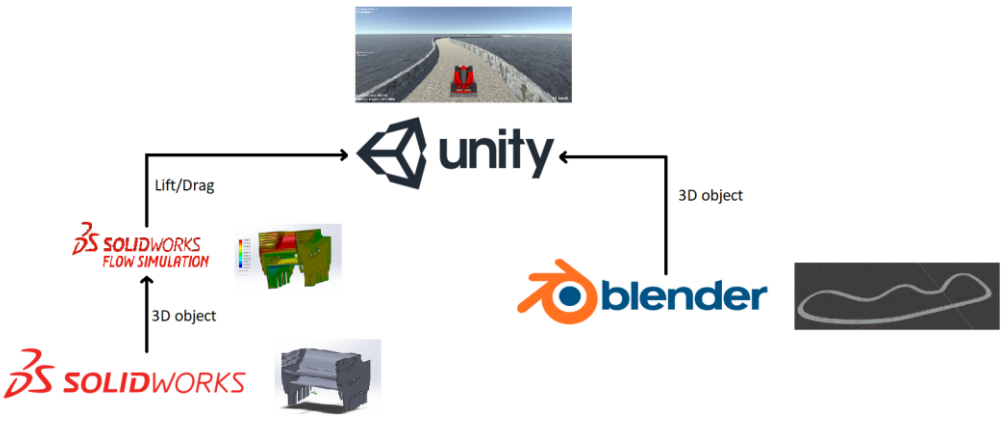

Fig. 1.1 Programs relations scheme [6], [7], [8]

2. THE COMPETITION

First of all, a brief introduction of the competition to help visualize the objectives of the car design.

This chapter explains the competition objectives, some background about the car's team, some detail about the competition challenges that the car will compete on and, finally, a brief explanation of the competition rules concerning aerodynamics.

Formula SAE (FSAE) (Society of Automotive Engineers), also called Formula Student, is a worldwide competition in which university students compete to design and build a formula-style single-seated competition car. FSAE competitions take place worldwide. Some host countries are the United States, the United Kingdom, Italy, etcetera. **(see Fig. 2.1).** Each country may have slight normative changes. This project will focus on the Formula Student Spain (FSS) competition hosted in Montmeló.

Fig. 2.1 Formula Student competitions in Europe [9]

2.1. Vilanova Formula Team

The car studied in this project is part of a Spanish team called Vilanova Formula Team (VFT).

This section introduces the team's background and how this project aims to help improve its performance.

VFT is a group of students from the Polytechnic School of Engineering of Vilanova i la Geltrú (EPSEVG) (I am the only student from outside the university) that started participating in Formula Student Spain in 2015. The first car was named *Nova 1*; since, the car has evolved to the newest version named the *Nova 6*.

Even though VFT is a relatively new team with low resources, the team has achieved decent results in later competitions. Despite a lack of improvement in the aerodynamics of the car. The first car with some thought into aerodynamics was the *Nova 6* **(see Fig. 2.2),** in which a team member called Joan Vandellós performed a TFG designing the rear wing and front wing of the car (see [10]), but due to time and budget limitations, it was never built.

This project will explain the design of the aerodynamics of the whole Nova 7, which hopefully will compete at the FSS 2022. This document will not discuss structural analysis, material selection and manufacturing. Its objective focus on how to improve car aerodynamics for future *Novas*.

Fig. 2.2 VFT racing car *Nova 6* [11]

2.2. Challenges

The car will compete in a series of challenges called events. Each event evaluates a different characteristic such as: acceleration, control or efficiency.

This section briefly describes the events of the competition, which will be interesting for the aerodynamic design. For more information, (see [12])

The competition events can be either static or dynamic. The events considered for this project are the Skidpad, Acceleration, Autocross, Endurance and Efficiency events (dynamic events). Each event will give points to each team according to their performance. All the dynamic events will be explained below. The overall maximum points achievable from these events are 525 and the static events sum up a total of 325 additional points **(see Table 2.1)**.

At the end of the weekend, the team who earned more points adding all the events wins the competition.

Table 2.1. Maximum points awarded for each event [12]

Static Events:	
Business Plan Presentation	75 points
Cost and Manufacturing	100 points
Engineering Design	150 points
Dynamic Events:	
Skid Pad	50 points
DV Skid Pad	75 points
Acceleration	50 points
DV Acceleration	75 points
Autocross	100 points
DV Autocross	-
Endurance	250 points
Efficiency	75 points
Trackdrive	-
Overall	1000 points

2.1.1. Skidpad

This event takes place in an “8” shaped circuit made of two pairs of concentric circles **(see Fig. 2.3)**. The driver will enter the circuit from the straight and will do two laps to the right circle, followed by two laps at the left circle. Only the second lap in each circle is timed. Therefore, the acceleration capabilities of the car are not too predominant. Aerodynamics in this event takes a crucial role as the additional downward force will allow it to turn faster.

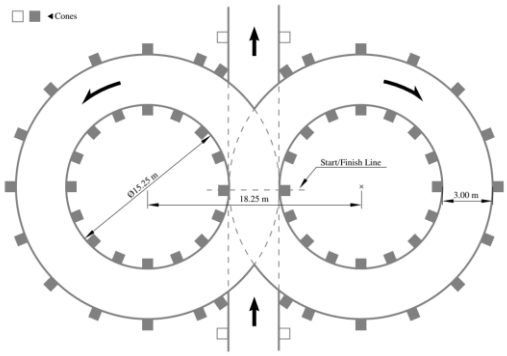

Fig. 2.3 Skidpad track layout [12]

Finishing the race awards the team with 3,5 points. Moreover, if the best time of the team (T_{team}) is lower than T_{max} (the best team’s time multiplied by 1,25), the team gets additional points based on the following formula:

$$M_{SKIDPAD_{SCORE}} = 46{,}5 \left(\frac{\left(\frac{T_{max}}{T_{team}}\right)^2 - 1}{0{,}5625} \right) \quad \textbf{(2.1)}$$

2.1.2. Acceleration

In this event, the car must accelerate from a standing start along a 75 m straight. Here it is very important to reduce the opposing horizontal force created by the air interacting with the car.

Finishing the race awards the team with 3,5 points. Moreover, if the best time of the team (T_{team}) is lower than T_{max} (the best team's time multiplied by 1,5), the team gets additional points based on the following formula:

$$M_{ACCELERATION_{SCORE}} = 46{,}5 \left(\frac{\frac{T_{max}}{T_{team}} - 1}{0{,}5} \right) \quad \textbf{(2.2)}$$

2.1.3. Autocross

This event takes place on a 1,5 km long track with an unspecified outline by the rules. The track can contain straights, constant turns, hairpin turns, slaloms, chicanes, etcetera. This year's competition track is jet to be known.

This event is the most challenging for aerodynamics as it combines straights with right and left turns at different speeds. The lap time simulator, developed in the last chapters of this project, will help to test the car's performance in this type of circuit.

Finishing the race awards the team with 4,5 points. Moreover, if the best time of the team (T_{team}) is lower than T_{max} (the best team's time multiplied by 1,25), the team gets additional points based on the following formula:

$$M_{AUTOCROSS_{SCORE}} = 95{,}5 \left(\frac{\frac{T_{max}}{T_{team}} - 1}{0{,}25} \right) \quad \textbf{(2.3)}$$

2.1.4. Endurance and efficiency

The Endurance event takes place in a closed lap circuit that follows the same guidelines as the Autocross event. The track is about 1 km, and the entire event is approximately 22 km.

Finishing the race awards the team 25 points. Moreover, if the best time of the team (T_{team}) is lower than T_{max} (the best team's time multiplied by 1,333), the team gets additional points based on the following formula:

$$M_{ENDURANCE_{SCORE}} = 300 \left(\frac{\frac{T_{max}}{T_{team}} - 1}{0{,}333} \right) \quad \textbf{(2.4)}$$

Additionally, during the endurance race, the Efficiency event also takes place. This Event measures fuel consumption, and therefore aerodynamics can also help reduce the volume of fuel burned. If the car is eligible for the Efficiency event, the team gets additional points based on the following formula:

$$M_{EFFICIENCY\_SCORE} = 75 \left(2 - \frac{EF_{team}}{EF_{min}} \right) \quad \text{where} \quad EF = T^2 \cdot E \quad \textbf{(2.5)}$$

2.3. Aerodynamic rules

This section briefly describes the more noteworthy aerodynamic rules of the competition. For more information about the rules (see [12]).

The main restrictions for aerodynamics are the keep-out-zones. It is not allowed to build aerodynamic devices in specific volumes like near the wheels **(see Fig. 2.4)**. This restricts the aerodynamic devices to be inside a generated volume **(see Fig. 2.5)**. Additionally, all aerodynamic devices must follow some security guidelines depending on their location in relation with the car. A more detailed explanation of some rules will appear when designing each car.

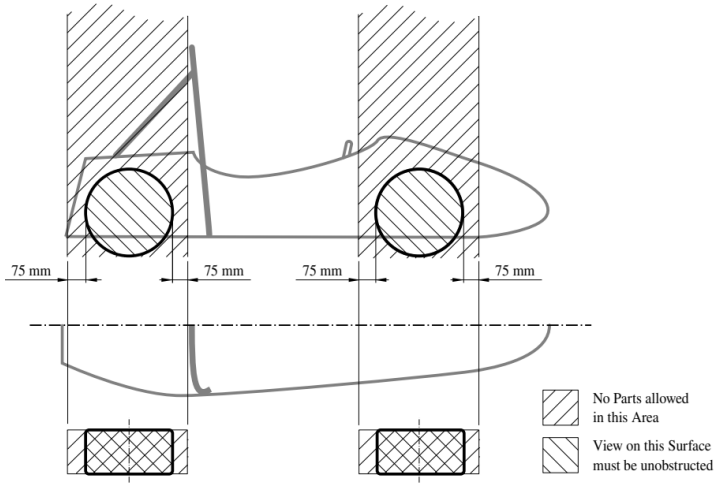

Fig. 2.4 Keep-out-zones for the definition of an open-wheel vehicle [12]

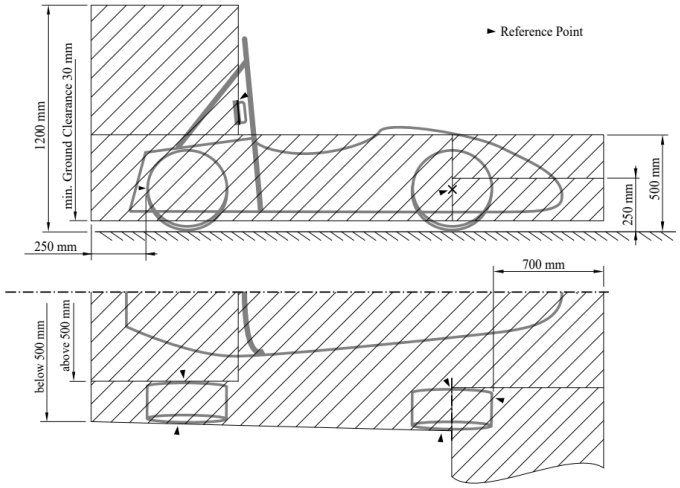

Fig. 2.5 Maximum dimensions and positioning of aerodynamic devices. [12]

3. AERODYNAMICS INTRODUCTION

Aerodynamics is the main subject of this project therefore some sort of introduction needs to be done to understand the base of the car's design.

This chapter explains the forces exerted by a moving object through a fluid and how are generated. This is the most basic knowledge about aerodynamics but is important to understand these concepts to understand the design of the aerodynamic devices. Moreover, this chapter explains how aerodynamics affects racing in a variety of ways.

The concepts explained in the first two sections are an adaptation of the fundamentals of aerodynamics book (see [13]).

When a fluid flows past an object or vice versa, its interaction makes some forces appear. These forces split into two perpendicular directions, the force perpendicular to the fluid direction is called Lift, and the force parallel to the fluid direction is called Drag. The science that studies this interaction when the fluid is air is called aerodynamics. Throughout this project, the fluid will be air, as it is the primary fluid that will surround the car.

3.1. Drag

Drag is the force exerted over the fluid direction. Usually, this force is undesired as it is opposite to the movement direction. Minimizing drag helps reduce fuel consumption or increase the maximum speed as the power required to counter the drag force will be less. The following fundamental aspects of drag generation help understand how to reduce the amount of drag produced.

Drag forces originate from two different stresses over the object's surface. Pressure distribution around the body creates pressure stress that acts perpendicular to the object's surface. Pressure stress depends mainly on fluid velocity. Shear stress, on the other side, acts tangential to the surface and is caused by the friction between the fluid and the object's surface. Shear stress depends mainly on the fluid's viscosity. Integrating both of them over the fluid's direction results in the drag force**.**

3.1.1. Pressure drag

Let's imagine a small volume of stationary air disturbed by a solid sphere flying across it. The air will first encounter the front part of the sphere, and this will force the air to surround it as it cannot travel through the inside. This interaction will compress the air particles at the front of the sphere, becoming a high-pressure area.

If the sphere velocity is high enough, the rear part will become a low-pressure area. The air displaced to the sides of the sphere will not be able to follow its shape and will detach the surface at a certain point, making the particle density lower. This low-pressure area will be called the separation region **(see Fig. 3.1).** The pressure difference will generate a force from the higher pressure area to the lower pressure area called pressure drag **(see Fig. 3.2).**

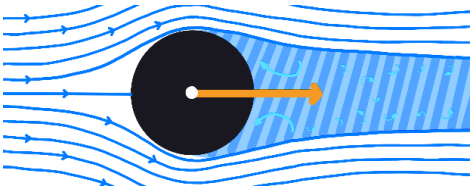

Fig. 3.1 Separation region created by a moving sphere

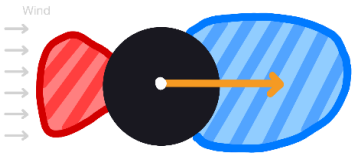

Fig. 3.2 High pressure area (red) and low pressure are (blue) around a moving sphere

Reducing the extension of the separation region will reduce pressure drag, instability, vortex shedding and unwanted vibrations. A turbulent boundary layer can remain attached to the surface better than a laminar one. Vortex generators are devices that transform laminar air into turbulent to reduce pressure drag.

3.1.2. Friction drag

In aerodynamics, most of the bodies are designed in a streamlined way to avoid a separation region and, therefore, reduce the pressure drag. But this also has a negative impact as the more streamlined a body is, the more surface it has and thus the more friction drag it will generate.

Flowing fluid over the surface, generates shear stress called friction drag. Unlike pressure drag, friction drag is higher if the boundary layer is turbulent. Therefore, the boundary layer type depends on which kind of drag is predominant **(see Fig. 3.3)**.

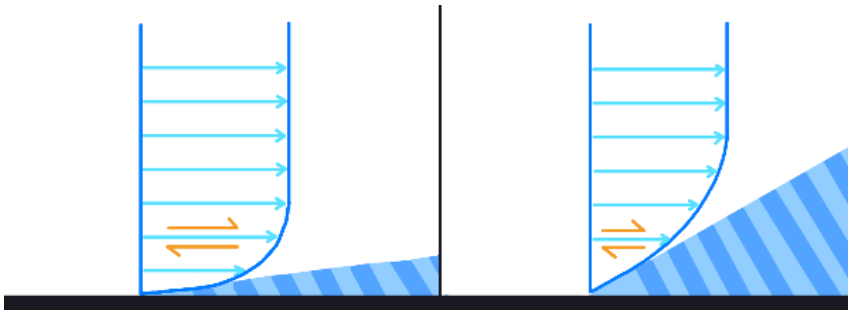

Fig. 3.3 Turbulent (Left) and laminar (Right) velocity profiles

In a nutshell, when streamlining a body, both types of drag are crucial to minimizing it **(see Fig. 3.4)**. It is necessary for straight lines, but when turning, lift is also essential, and it will not be possible to achieve minimum drag every time. The nose cone and bodywork are the main elements that focus on drag minimization.

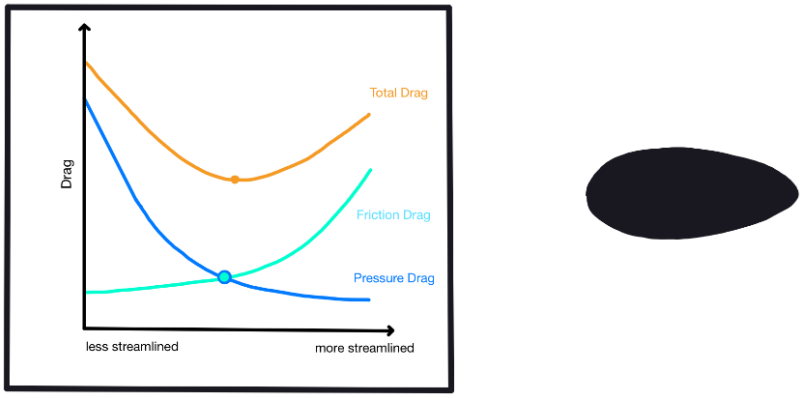

Fig. 3.4 Drag vs object shape plot (Left) and minimum drag airfoil (Right)

3.2. Lift

Lift is the force exerted perpendicular to the fluid direction. Lift in racing allows the car to corner at higher velocities (further explained in Importance of aerodynamics in racing section), so the objective is to maximize it.

Lift is a product of the same two stresses (shear and pressure stress), but integrated over the perpendicular direction. For streamlined bodies like airfoils, shear stresses will mainly be acting in the same direction as the flow. Thus they will not contribute significantly to lift. Therefore, pressure distribution around the object is the main lift generator. Bernoulli's principle or Newton's third law helps to understand pressure distribution.

3.2.1. Bernoulli’s principle

Bernoulli's principle states that when the velocity of a fluid increases, its pressure reduces, which is just a statement of the conservation of energy. In a racing car, the objective will be to accelerate the air passing below the object to create a lift force pointing to the ground. This concept is key to understanding how the car’s undertray works.

Why fluid is accelerated is not very intuitive. Imagine two air particles that encounter a moving airfoil. One particle will travel above the airfoil and the other one below it. If the airfoil is asymmetric, one particle will travel a longer distance before reaching the trailing edge (rear part of an airfoil) than the other. Therefore, if both particles travel at the same velocity, they should travel the same distance and encounter at a physically impossible point **(see Fig. 3.5).**

Fig. 3.5 Airfoil streamlines without circulation

Kutta’s condition solves this by stating that, to achieve parallel flow at the trailing edge, the air travelling the longest distance needs to be accelerated, and the air travelling the shorter distance needs to be deaccelerated thanks to circulation **(see Fig. 3.6).**

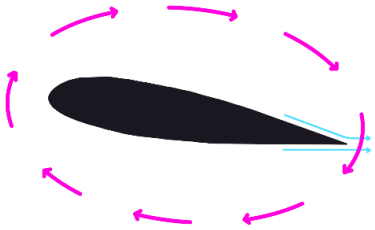

Fig. 3.6 Circulation around an airfoil

3.2.2. Newton’s third law

Newton's third law states that, for every action there is an equal and opposite reaction.

Steady air before interacting with an airfoil has a horizontal movement relative to the airfoil. After leaving the trailing edge, the air has been redirected upward (upwash) or downward (downwash), depending on the geometry of the airfoil.

Applying Newton's third law, diverting the air upward will create an opposite force pointing to the ground. This simple physic concept is essential for designing the car's diffuser.

3.3. Importance of aerodynamics in racing

After explaining the aerodynamic forces generated over a moving device and how can they be generated, an explanation of how these can be related to racing is necessary.

This short section explains how each aerodynamic force affects a racing car and whether they are useful or not.

A racing track is composed by straights and turns of different lengths and angles. During a straight, the only forces acting on the car are in the direction the car is moving. In this case the desired configuration is to have the less drag possible as it exerts an opposite force to the engine acceleration. Therefore, drag affects acceleration and maximum speeds.

On the other hand, when the car is turning, lateral forces appear because of the centripetal forces **(see Fig. 3.7).** These centripetal forces pull the car outside the circle and increase with speed. To counteract these forces, the car tires, exert friction with the ground, and as long as this friction is higher than the centripetal forces the car will not slip.

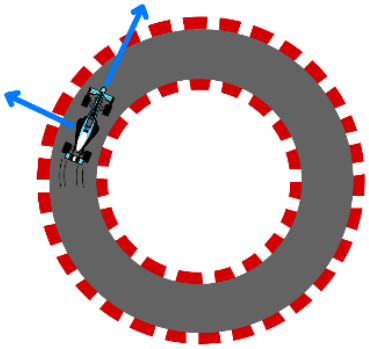

Fig. 3.7 Centripetal and acceleration forces on a turning car

The friction generated when turning depends mostly on the tires, the car weight and the aerodynamic downforce. Therefore, increasing lift allows the car to turn faster with the same turning angle without skidding.

In a nutshell, low drag is needed for straights and high downforce is needed for turns. But drag and lift are correlated, and because of it, the perfect balance will depend on the number of straights and corners the circuit has.

3.4. Racing line

In relation to the previous section, an explanation of the car's trajectory will help to understand the need for a deep learning algorithm instead of creating a fixed trajectory for the lap time simulator.

This section explains why the car's trajectory varies for the same type of curve depending on the following track section.

A racing track has a certain width that allows the cars to make their path inside the track. They will try to do the perfect path to beat the circuit in the shortest time possible. This path is called the racing line and they are not easy to calculate.

In a simple corner, the fastest path possible is the one generated by a circle that goes from the outside of the track, touches the inside right at the middle of the curve (geometrical apex) and ends at the outside part of the track again **(see Fig. 3.8).** The velocity at which the car will be able to do it depends, among other factors, on aerodynamics.

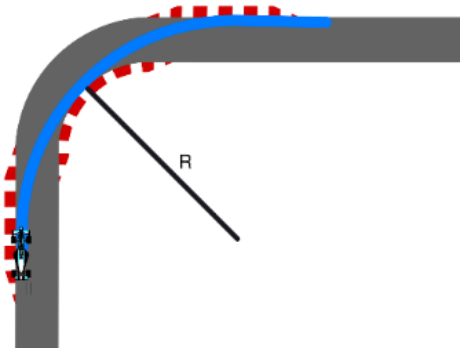

Fig. 3.8 Geometrical apex turn of radius R

This path explained optimizes only a particular turn, but the racing line needs to evaluate the whole circuit and therefore it is important to know what comes after the curve.

If there is a long straight, the perfect line will be to brake later, and harder, turning later. This will allow the driver to accelerate earlier and reach higher velocity during the long straight. As the inside of the curve is reached later than the geometrical apex, this is called the late apex **(see Fig. 3.9).**

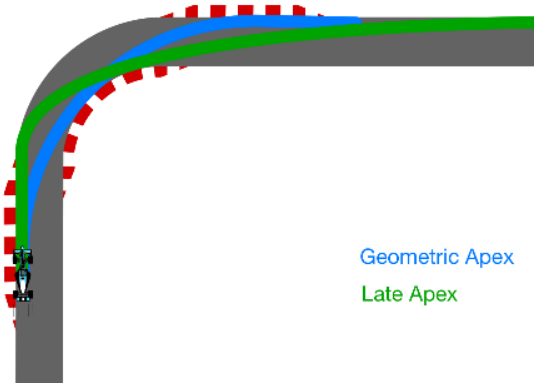

Fig. 3.9 Geometric apex vs late apex

If the curve is followed by a slow turn, the strategy will be to maximize the entry speed by doing an early apex and therefore sacrificing the exit speed **(see Fig. 3.10)**.

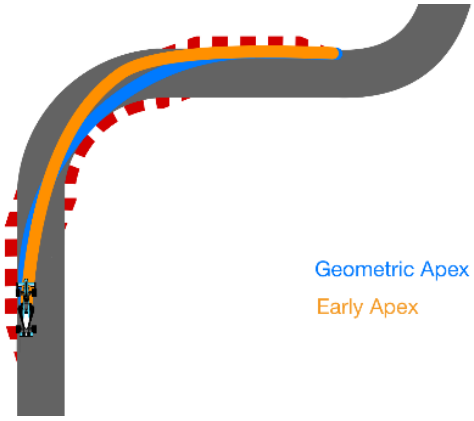

Fig. 3.10 Geometric apex vs early apex

Racing lines are not easy to calculate as they may vary from car to car or even during the race because of weather conditions, tire deterioration or fuel consumption. In this project, the racing line will be calculated by the lap time simulator using the car aerodynamics designed and simulated by CFD.

4. ALGORITHMS

Having introduced all the aerodynamic background, the algorithms used for the lap time simulator need some explanation too. These algorithms will work together to find the optimum path and velocities to achieve the best lap time with a determined car and track.

This chapter contains a general explanation of what is a neural network and a genetic algorithm and how they work together. The actual implementation will be explained in a later chapter.

4.1. Neural network

This section shows how a neural network works and how it can be configured to obtain an adequate network for this project. The contents in this chapter are an adaptation from the first chapter of the book: An introduction to neural networks (see [14]).

Neural networks can be understood as “computer’s brain”. The human brain is complex, a part of its work is to generate an output based on an input. Let’s imagine, for example, a football goalkeeper. The goalkeeper tries to intercept a ball before it reaches the goal net. The images the brain receives through the eyes are the inputs, and the brain processes this information and moves the goalkeeper's body (outputs) according to this information.

Neural networks are used in several ambits like facial recognition, handwriting recognition or stock market prediction. In this project, the neural network will be used as the “brain” of a racing car pilot. The neural network will receive some inputs from the car and return some outputs such as the acceleration or the steering wheel direction. But before entering more detail, let’s explain how neural networks work with a simple example.

Imagine a field with two types of flowers, one with red petals and the other one with blue petals. Someone has counted the number of petals of each flower and the stem’s length. The neural network must identify if the flower is red or blue based only on these two attributes. Therefore, the neural network will have two inputs (number of petals and stem length) and two outputs (likelihood of being red or blue).

First, let’s create a simple network for this case by connecting each output neuron with both input neurons **(see Fig. 4.1)**. Each connection has a value associated called weight and measures its strength. Therefore, each output will be the sum of the connected inputs multiplied by its respective weight **(eq. 4.1 and 4.2)**.

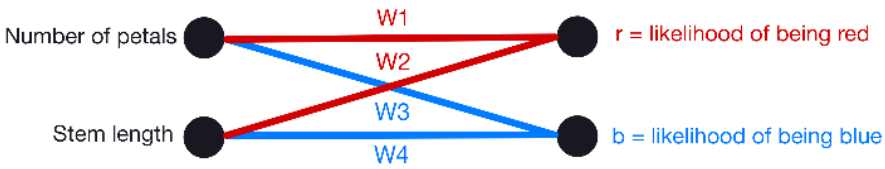

Fig. 4.1 Neural network with 2 inputs and 2 outputs

$$r = Number\ of\ petals \cdot w_1 + Stem\ Length \cdot w_2 \quad \textbf{(4.1)}$$
$$b = Number\ of\ petals \cdot w_3 + Stem\ Length \cdot w_4 \quad \textbf{(4.2)}$$

The weight values are the variables that need to be adjusted to obtain valid outputs. But this network is too simple for this case. First, plot the flower attributes in a two-dimensional chart and draw a line when red likelihood equals blue likelihood **(see Fig. 4.2)**. Adjusting the weights to separate both clusters of flowers is impossible with this network because the separating line will always cross the chart's origin. This is why a constant called bias will be added to each output equation **(see Fig. 4.3) (eq. 4.3 and 4.4)**.

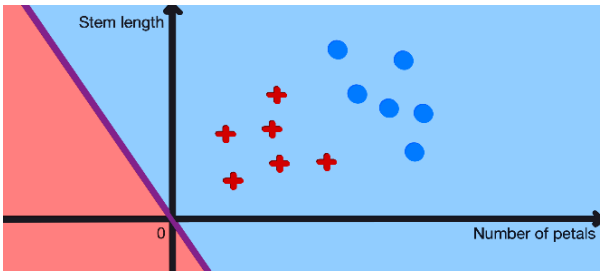

Fig. 4.2 Flower's sample attributes plotted with linear separation line (no bias)

$$r = Number\ of\ petals \cdot w_1 + Stem\ Length \cdot w_2 + b_1 \quad \textbf{(4.3)}$$
$$b = Number\ of\ petals \cdot w_3 + Stem\ Length \cdot w_4 + b_2 \quad \textbf{(4.4)}$$

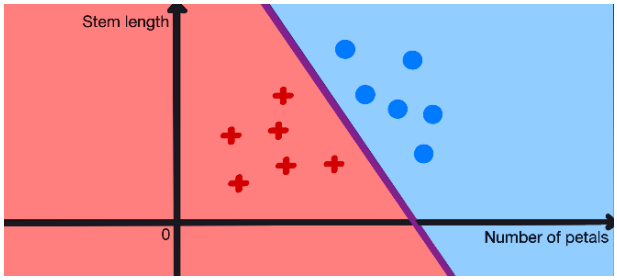

Fig. 4.3 Flower's sample attributes plotted with linear separation line (with bias)

This simple network will work correctly for simple cases where a straight line can separate the two clusters. But, to separate more complex samples, the network needs to have more weight variables and introduce non-linearity. Notice that more weights mean more flexibility and more variables to tune. Therefore, the complexity of a neural network will depend on each situation. This tuning is the most challenging part of creating a neural network.

Adding a new layer called the "Hidden layer" with three hidden neurons to the prior example will introduce more weights **(see Fig. 4.4).** With this new layer, the output equation will be more complex, but will remain linear. Applying a sigmoid function such as the logistic curve **(see Fig. 4.5)** to the hidden and output layers will introduce non-linearity. Moreover, it encloses the value between 0 and 1. To use negative values, a hyperbolic tangent can be used instead.

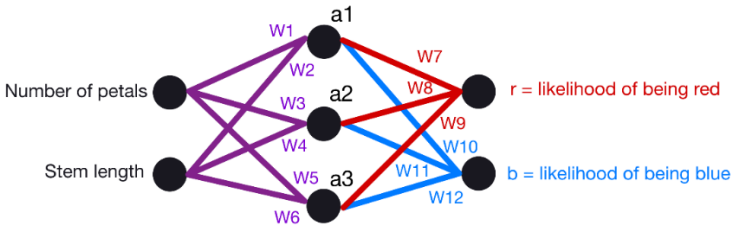

Fig. 4.4 Neural network with 2 inputs 3 hidden neurons and 2 outputs

$$a_1 = Number\ of\ petals \cdot w_1 + Stem\ Length \cdot w_2 + b_1 \quad \textbf{(4.5)}$$

$$a_2 = Number\ of\ petals \cdot w_3 + Stem\ Length \cdot w_4 + b_2 \quad \textbf{(4.6)}$$

$$a_3 = Number\ of\ petals \cdot w_5 + Stem\ Length \cdot w_6 + b_3 \quad \textbf{(4.7)}$$

$$r = a_1 \cdot w_7 + a_2 \cdot w_8 + a_3 \cdot w_9 + b_4 \quad \textbf{(4.8)}$$

$$b = a_1 \cdot w_{10} + a_2 \cdot w_{11} + a_3 \cdot w_{12} + b_4 \quad \textbf{(4.9)}$$

$$f(x) = \frac{1}{1+e^{-x}} \quad \textbf{(4.10)}$$

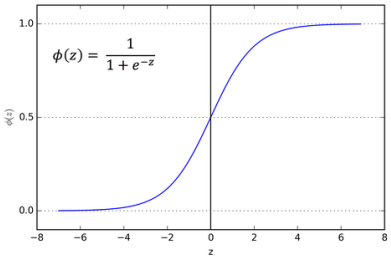

Fig. 4.5 The logistic curve [15]

$$a_1 = f(Number\ of\ petals \cdot w_1 + Stem\ Length \cdot w_2 + b_1) \quad \textbf{(4.11)}$$

$$a_2 = f(Number\ of\ petals \cdot w_3 + Stem\ Length \cdot w_4 + b_2) \quad \textbf{(4.12)}$$

$$a_3 = f(Number\ of\ petals \cdot w_5 + Stem\ Length \cdot w_6 + b_3) \quad \textbf{(4.13)}$$

$$r = f(a_1 \cdot w_7 + a_2 \cdot w_8 + a_3 \cdot w_9 + b_4) \quad \textbf{(4.14)}$$

$$b = f(a_1 \cdot w_{10} + a_2 \cdot w_{11} + a_3 \cdot w_{12} + b_4) \quad \textbf{(4.15)}$$

With this, the neural network will be able to divide a great variety of clusters and therefore identify if a flower is either red or blue just by its number of petals and its stem's length **(see Fig. 4.6)**.

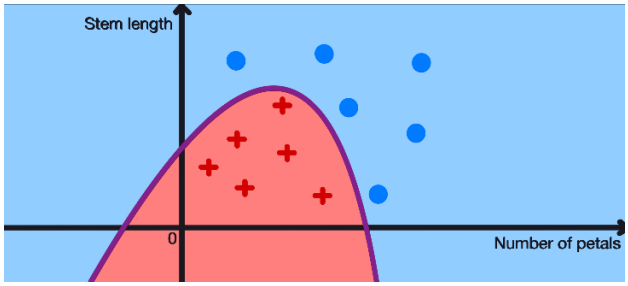

Fig. 4.6 Flower's sample attributes plotted with non-linear separation line

In a nutshell, neural networks work as a very complex function that gives outputs based on inputs. The network consists of neurons and connections. The number of neurons is usually fixed, but weights and biases do. Tunning these weights can be tedious to do manually, and here is where genetic algorithms come into play.

4.2. Genetic algorithm

This section explains how genetic algorithms work and how they will be used to obtain the best neural network. This section is an adaptation of "Introduction to Genetic Algorithms for Combinational Problems" by Miguel Valero, UPC (see [16]).

Charles Darwin's theory of natural evolution inspires genetic algorithms. This theory states that the fittest individuals will survive and reproduce between them, improving the genetics of the next generation. This concept can be applied to solve a great variety of problems, and in this project, it will be used to create the best neural network (i.e. the best pilot's brain for a particular circuit).

4.2.1. Initial population

First, the algorithm will randomly generate, a set of neural networks (individuals). This set of individuals is called the population, and each can be a possible solution. An array with all the weight values will represent each neural network (chromosome). This array will be called a chromosome, and each weight value will be a gene **(see Fig. 4.7).** Every neural network will have the same architecture of neurons and links. The only difference between them will be the weight values.

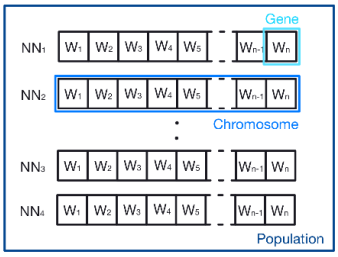

Fig. 4.7 Gene, chromosome and population definitions

4.2.2. Fitness function

To select the best individuals, each of them receives a score which will determine how fit they are. The fitness function is used to calculate this score by evaluating the results achieved by each individual. As the objective is to find the best neural network that can drive a racing car, the fitness function will evaluate parameters such as: how fast the car is and how long it can travel before crashing.

4.2.3. Selection

After assigning a fitness score to each individual, the fittest are selected and will pass to the next generation. Moreover, sets of pairs will be made from this selection for later reproduction. The more fit an individual is, the more probability it has to be selected and, therefore, the more times it will be used for reproduction.

4.2.4. Crossover

The previously selected pairs mix their genes, creating two new chromosomes **(see Fig. 4.8)**. There are lots of ways to mix the genes during the crossover. The simplest is to split the chromosome in half and switch the second half between the pair.

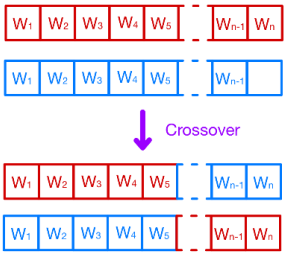

Fig. 4.8 Simple crossover between two chromosomes

4.2.5. Mutation

With the four previous steps, the algorithm will evolve and converge, but, most certainly, the final solution will not be the best one possible. This is because the only randomness applied to search for better individuals is by generating a random chromosome during repopulation and mixing genes from the fittest during the crossover. But to be sure of the solution, the algorithm needs to explore more variations, and here is where mutation comes into play. Mutation chooses some random genes of the chromosomes created during the crossover and changes them to an entirely new random gene.

4.2.6. Repopulation

All the chromosomes result from the steps before proceeding to the next generation. This new generation is also filled with randomly generated individuals until the population has the same size as the initial population. After, the cycle repeats (skipping the initial population step) until it reaches the desired goal **(see Fig. 4.9)**. In the pilot's example, the goal can be to complete a lap underneath a specific time.

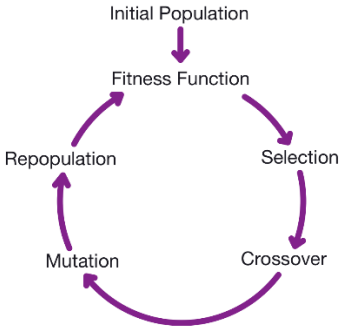

Fig. 4.9 Genetic algorithms loop

4.3. Algorithms applied

Finally, after a brief introduction of how a neural network and a genetic algorithm work, let's explain how they are implemented in this project to make the lap time simulator.

This section is a brief introduction to the use of both algorithms. The actual implementation will be explained in detail later on.

First of all, a random neural network is created. The neural network is fully responsible for the car movement and because the first one is randomly generated it will probably be a bad driver. The network is tested by making it run a car inside a defined circuit until it crashes and then it is given a score to evaluate how well it has performed.

The process of creating a random network and evaluating its performance is repeated several times. Up to this moment the only algorithm applied is the neural network. The genetic algorithm is applied after testing the initial set of networks (initial population).

The genetic algorithm takes a set of neural networks and initiates the process of selection, crossover, mutation and finally repopulation **(see Fig. 4.9)** . First, the networks are selected depending on their fitness score during the testing. Then, the selected networks are reproduced together during the crossover and the resultant children are saved for the next generation along with the best selected networks. Finally, after some randomness is applied by using mutation, the next generation is filled with new randomly generated networks.

By repeating this process, the networks will improve their performance scores over generations until finally become good drivers.

5. PLANNING

Now that the problem has been introduced and the theoretical background has been explained, it is time to define the goals of the project and define a work plan to achieve these goals.

5.1. Objectives

The main goal of the project is to design the aerodynamics of a Formula SAE car and create a tool to test it. This main goal can be divided into designing and testing:

5.1.1. Designing the car

To achieve the goal, the designed car must:

- Follow the rules of the competition (see [12]).
- Improve the aerodynamics of the past season's car (*Nova 6*).
- Be feasible to build (i.e. the car design should not include elements that cannot be built in real life).
- Be compatible with the designs of other departments (e.g. the bodywork must cover the chassis designed by the dynamics team).
- Be validated by CFD simulations.

5.1.2. Creating the testing tool

To achieve this goal, the testing tool must:

- Use genetic algorithms to reduce the testing time.
- Use an optimized neural network (i.e. the network parameters should be modified to reduce the simulating time).
- Use realistic physics.
- Achieve results better than a non-professional pilot.
- Be able to compare the lap times of different aerodynamic configurations.

5.2. Work plan

Having established the objectives of the project, the work plan can be specified. This section describes the different processes and time scheduled for each task.

The first thing to do is mesh refinement. This needs to be completed before doing any simulations and this is why the earlier it is done, the earlier the simulations can start. In the following chapter, there is an explanation of the mesh refinement.

Parallel to the mesh refinement the 3D objects of the aerodynamic devices can be designed. Thanks to the prior experience in a formula team, this task can be finished before the mesh refinement.

Once the two previous tasks are done, it is time to do the CFD simulations. This task cannot finish faster than what the CPU needs to complete them. Therefore, this task should be started as soon as the two previous ones are done.

Parallel to the aerodynamics work, the lap time simulator can be developed. The first thing is to develop a self-driving car with simple car movements to develop just the algorithms.

Once the self-driving car is completely functional and tested, advanced physics can be coded to make the simulation more realistic. This task would take lots of research. Therefore, the research should be done parallel to the aerodynamic task to later expend time exclusively on the coding and implementation.

The last task of the project is implementing both parts together. To do so, the CFD simulations need to be finished and the lap time simulator should be already optimized. Then, the final conclusions of the project can be made.

Note that the Gantt diagram of this project **(see Fig. 5.1)** has been modified to match the actual duration of each task.

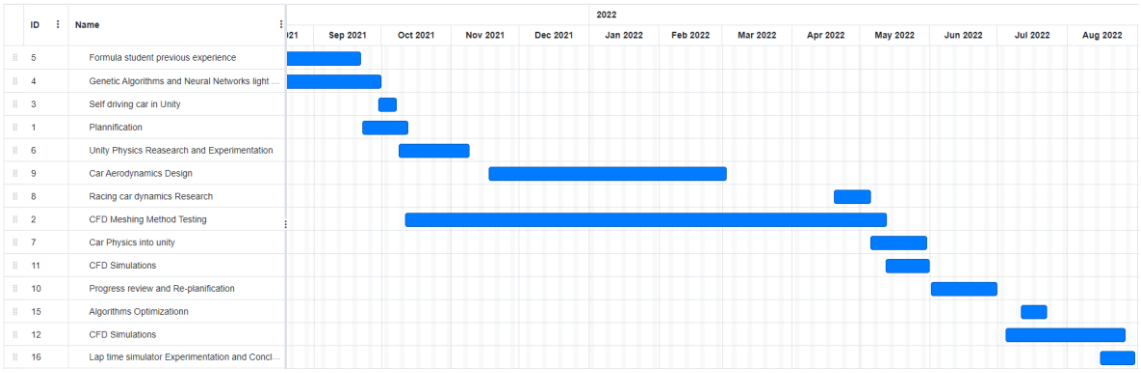

Fig. 5.1 Project Gantt diagram over 1 year [17]

6. CFD METHODOLOGY

After the fundamental theory has been explained and the project has been set, the way a CFD solver works needs to be understood before the actual development of the project starts.

Every fluid behavior can be predicted by using a set of equations called Navier Stokes equations. These equations cannot be solved, but using a variety of techniques Computational Fluid Dynamics (CFD) can solve approximations to the equations.

This chapter contains the explanation of what is CFD, how to configure the solver and the mesh refinement study that will be used throughout the project.

References consulted for this chapter: [18], [19], [20], [21] and [22].

6.1. Navier Stokes and CFD

First, let's start with an explanation of how a fluid behaves and how it can be analyzed mathematically. This section explains what are exactly the Navier Stokes equations and what approximations will be used by CFD to solve them.

All the fluids of the project will be assumed to be Newtonian, Incompressible and Isothermal.

A Newtonian fluid means that shear stress rate has no effect on the fluid's viscosity. A good example of how a non-Newtonian fluid behaves would be a bottle of ketchup. When the bottle is upside down the ketchup's viscosity makes the fluid stick to the bottle. But when applying shear stress to the bottle by hitting it, the viscosity of the fluid is reduced and exits the bottle. For a Newtonian fluid the viscosity would have remained constant. Example from: (see [23]).

An incompressible fluid does not change its volume when it is being compress at pressure. This means that the fluid density is constant.

An isothermal fluid does not lose energy in the form of heat.

These three assumptions are valid for the aerodynamics of our car and by using them the problem is being simplified significantly. These assumptions take to the Navier Stokes equations. They are based on well-known physics properties. The first equation states that mass is conserved **(see eq. 6.1)** and the second one is a version of Newton's second law **(see eq. 6.2).**

$$\nabla \cdot u = 0 \qquad \textbf{(6.1)}$$

$$\rho \frac{du}{dt} = -\nabla p + \mu \nabla^2 u + F \qquad \textbf{(6.2)}$$

The mass conservation equation states that the divergence of the velocity vector field of the fluid equals zero. The divergence of a vector field tells how much a point tends to divert vectors away from it. Therefore, a negative divergence diverts vectors to the point and a positive divergence away from the point **(see Fig. 6.1).**

When talking about fluids, a divergence would act like a source generating or destroying the fluid. Therefore, the divergence has to be zero because of mass conservation.

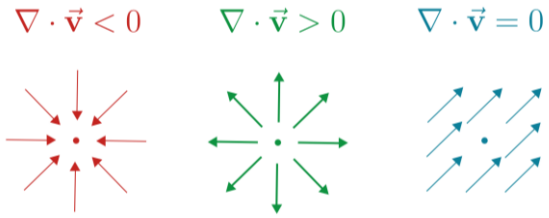

Fig. 6.1 Flow vectors for positive (Left), negative (Middle) and neutral divergence (Right) [24]

Newton's second law states that the sum of forces acting on a body equals its mass times the acceleration.

$$\mathrm{ma} = \sum F \qquad \textbf{(6.3)}$$

The Navier Stokes equations are applied to individual points. Therefore, **eq. 6.3** needs to be divided by volume resulting in **eq. 6.4.**

$$\rho \mathrm{a} = \sum F \qquad \textbf{(6.3)}$$

Next, acceleration can be replaced by the derivative of the velocity vector field resulting in **eq. 6.5.**

$$\rho \frac{du}{dt} = \sum F \qquad \textbf{(6.3)}$$

The forces can be split into internal and external forces. Internal forces are exerted by the molecule on the surrounding fluid and external forces are the ones exerted on the molecule by some external object.

The first internal force is pressure. As explained in AERODYNAMICS INTRODUCTION, a pressure difference creates a force from the high pressure area to the low pressure area. This is represented in the equation as a negative pressure gradient ($-\nabla p$).

The second force is viscosity or friction. This is a property of the fluid and can be represented as $\mu\nabla^2 u$.

The external forces are represented by the letter F in **eq. 6.2**, but it is pretty common that gravity is the only internal force (pg).

Theoretically this Navier Stokes equations can predict the future of a fluid from a current state, but turbulence makes the fluid chaotic and therefore it is incredibly hard to make long term predictions. This is why models of turbulence are used in CFD.

6.2. Turbulence models

Over the years, many different methods were invented to model the behavior of turbulent flows. Some examples are: Direct Numerical Simulation (DNS), Turbulent-viscosity models, Reynolds-stress Models (RSM), Probability Density Function methods (PDF) and Large-Eddy Simulation (LES).

Solidworks Flow Simulation uses a Turbulent-viscosity model called the k-ε model[1] with some additional empirical enhancements like the damping functions proposed by Lam and Bremhost.

The Turbulent-viscosity models use the Reynolds-averaged Navier Stokes (RANS) equation which uses the time-averaged velocity field. They are based on the Turbulent-viscosity hypothesis[2].

Other Turbulent viscosity models are: Spalart-Allmaras (S-A) or the k-omega (k-ω).

6.3. Initial conditions and boundary conditions

After introducing how Navier Stokes equations can be used to study the behavior of a fluid and the need of a turbulence model, there are two more conditions that need to be defined to solve the problem. These are the initial conditions and the boundary conditions.

[1] For a detailed explanation refer to [20] pg. 373-383
[2] For a detailed explanation refer to [20] pg. 359

This section explains the importance of the initial conditions and boundary conditions in a CFD solver and specified the parameters that will be used for all the simulations in this project.

The initial conditions are used to determine all the variables at the time t=0. Because the aim of this project is to design the aerodynamics for a car, the simulations are stationary. This is, the fluid velocity will not change over time and the fluid density, pressure and temperature will remain constant.

The boundary conditions connect the simulation model with its surroundings and are active throughout the simulation, unlike the initial conditions. But, as the simulations are stationary, the boundary conditions will remain constant.

The initial conditions **(see Table 6.1)** are set to the standard atmosphere and ideal walls without roughness or thermal propagation. The car will be made mostly of carbon fiber, which has low roughness and thermal propagation at sub Mach speeds. Finally, velocity is set to 50 km/h, the average the car will be at when turning.

The boundary conditions are set to a velocity inlet from where the flow will enter the domain, three slip walls, a pressure outlet and a symmetry condition **(see Fig. 6.2**). The slip walls need to be far enough to not alter the flow trajectory (slip walls do not alter the tangential flow but they restrict normal flow). And the symmetry condition is set to a plane where the normal velocity is zero to accelerate the simulations reducing by half the number of cells. Moreover, for simulations close to the ground, an additional moving wall will be added with the same velocity and direction of the inlet flow.

Table 6.1 Initial conditions

INITIAL CONDITIONS	
Fluid density	Air (1.225 kg/m3)
Velocity	50 km/h
Wall thermal condition	Adiabatic Wall
Roughness	null
Pressure	101325 Pa
Temperature	293.2 K

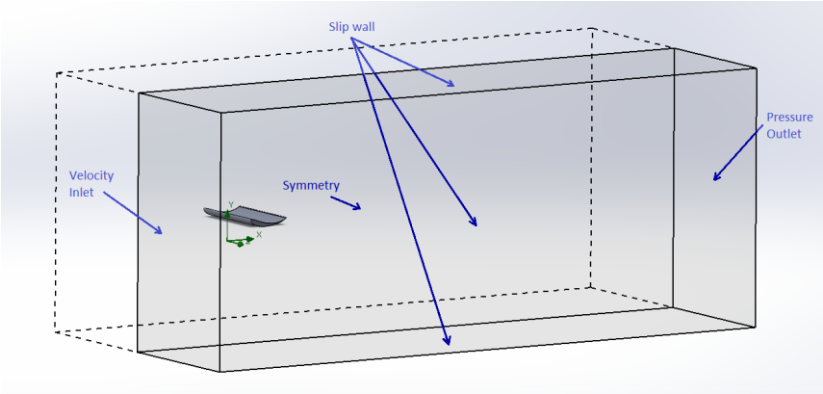

Fig. 6.2 Computational domain boundary conditions

For the following sections and chapters, the car will be pointing opposite the x-axis with the y-axis pointing upwards and, therefore, the z-axis pointing to the left side **(see Fig. 6.3).** Hence, the air will be in the x-axis direction.

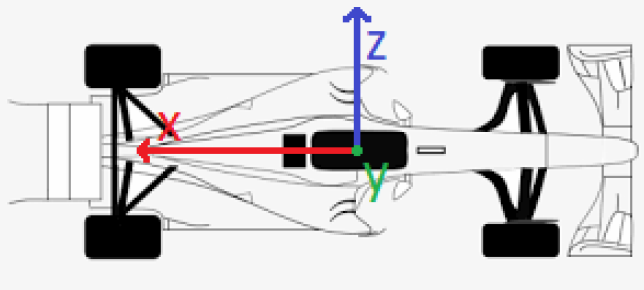

Fig. 6.3 Car orientation [25]

6.4. Mesh

Finally, before making any CFD simulation comes the mesh refinement, and to understand what this means first, let's explain what a mesh is. This section shows what is the mesh in a CFD simulation and the configuration that will be used throughout the project.

In the real world, millions of particles are inside defined air volume. Ideally, a CFD simulation should calculate how each of these particles moves and interact with each other, but that is impossible nowadays. Instead, splitting the volume into cells, the calculations only occur on the vertex or the center of each cell. This volume formed by multiple cells is called mesh, and mesh refinement consists in finding the optimal mesh. A too fine mesh increases the computation time enormously, but a too coarse one can lead to inaccurate results.

The first step is to define the computational domain, which is the volume considered by the simulation. The solid used throughout this section is an MSHD (see [1]), a high lift airfoil used for the rearwing. The airfoil has a length of 400 mm and a width of 860 mm. As a general rule, the computational domain will be longer along the horizontal direction and with more space behind the solid to simulate how the air affected by the solid behaves. So, the computational domain will have a size of 6 x 2,5 x 3 m and the solid will be centered at the x-axis but leaving only 1 of the 6 m at the front **(see Fig. 6.4).**

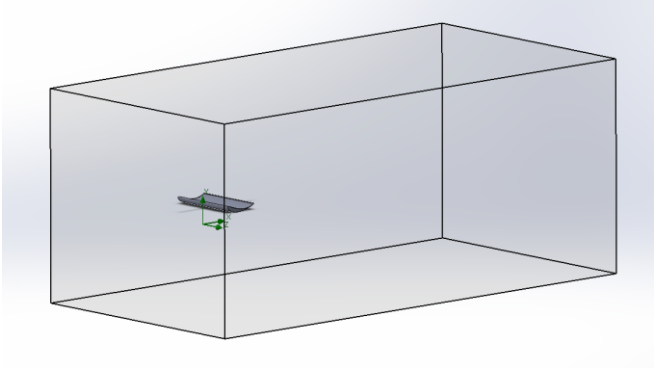

Fig. 6.4 MSHD with a 6 x 2,5 x 3 m computational domain

Solidworks generates the mesh inside the computational domain in accordance with specified parameters. First of all, it generates a global mesh. It can be modified using two main parameters, the quality level and the ratio factor. The quality level automatically tunes more specific parameters to determine the size of each cell depending on the solid and computational domain size (later on, when refining the mesh for a new object, these parameters will be copied manually to have consistency when comparing global mesh qualities). The ratio factor indicates how the cell's size increase when moving away from the solid **(see Fig. 6.5)**, meaning that a ratio factor of 1 will have all the cells equally sized. The ratio factor is useful because the air near the solid will be much more affected than the air far away.

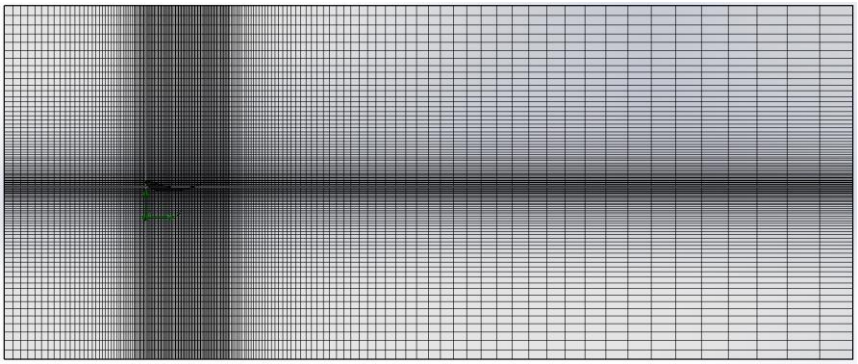

Fig. 6.5 Lateral view of a Global Mesh with ratio factor 1,2

Solidworks can do some local refinement thanks to a local mesh that divides cells from the global mesh. One method is to create the local mesh with an adaptive solution that refines the mesh only where needed. This method obtains good results for an individual simulation but makes harder the comparison of different objects' solutions as each object will have a different mesh.

Therefore, the local mesh is created using equidistant mesh refinement. The surroundings of the solid are divided into different equidistant shells (3D volume with constant thicknesses around the solid) with different thickness each **(see Fig. 6.6).** First, each cell inside the volume between the solid and the biggest shell is split into 8 smaller cells. Then, the cells inside the next smaller shell are split again. This process is repeated until the smallest shell is reached.

By using this local mesh method, with a 2 shell configuration, the cells around the solid will be 64 times smaller than the global mesh cells and the cells inside the second shell will be 8 times smaller than the global mesh cells.

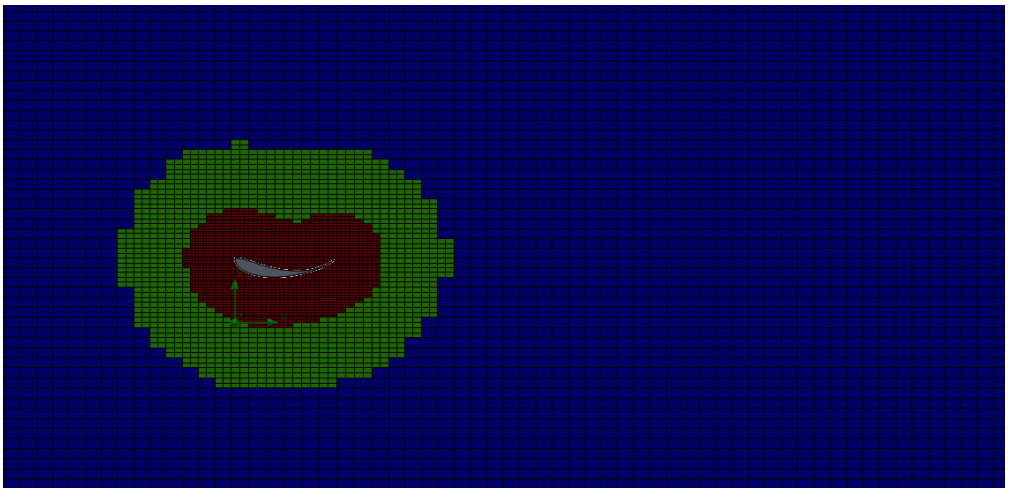

Fig. 6.6 Lateral view of local mesh with 2 equidistant shells

6.4.1. Mesh refinement

The following table **(see Table 6.2)** (full results in Table A.1) shows the different results varying the number of local mesh shells, the local mesh quality, the global mesh quality and the ratio factor, that resulted in a reasonable number of cells for the simulation (less than 14 million).

Table 6.2 MSHD initial mesh refinement results (simplified and sorted by ascendant number of mesh cells)

	Mesh cells	CPU Time	Fx (Drag)	Fy (Lift)
MSHD(9)	777.533	0:15:11	1,915	-4,292
MSHD(6)	1.554.694	0:52:08	1,77	-6,548
MSHD(11)	1.809.216	0:49:28	1,792	-3,14
MSHD(10)	2.018.027	0:51:10	1,926	-4,515
MSHD(1)	2.144.029	3:09:50	2,384	-5,779
MSHD(3)	2.458.378	1:59:56	1,831	-5,681
MSHD(13)	3.069.124	1:43:55	1,632	-5,959
MSHD(7)	4.412.766	3:23:40	1,689	-6,323
MSHD(12)	4.765.855	2:28:21	1,702	-3,607
MSHD(4)	4.930.141	5:04:43	1,784	-7,915
MSHD(15)	6.232.782	4:37:26	1,753	-4,925
MSHD(14)	8.108.613	5:40:52	1,674	-5,174
MSHD	8.593.560	44:57:23	2,81	-9,282
MSHD(8)	10.677.626	10:09:07	1,691	-6,307
MSHD(5)	10.803.184	16:19:11	1,935	-8,91
MSHD(2)	14.569.033	38:54:01	2,014	-7,541

Some tests required a very high number of iterations to complete resulting in an unreasonable amount of CPU time. After some experimentation, the conclusion was that sometimes the results oscillated without reaching the required delta and lead to errors in the result. That is why these results should be discarded (MSHD and MSHD (2) are perfect examples of this).

From the table and some visual aid, the best shell size determined is the selected for shell 2 (smallest shell) of MSHD(8)**,** as by looking at the Mach number plot **(see Fig. 6.7)**, most of the action happens within the shell. Moreover, shell 3 is unnecessarily large. Finally, the ratio factor selected is adequate for this case as the gradients are mainly vertical.

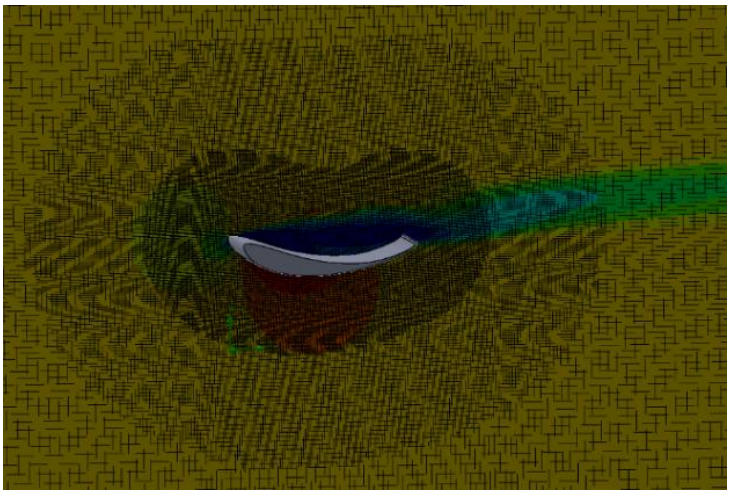

Fig. 6.7 Close up view of MSHD(8) with Mach plot

From the new variations of the MSHD(8) **(see Table 6.3)** (full results in Table A.2)**,** after discarding some results with prohibitive amounts of CPU time, resulted in four valid simulations (highligted in green on the table). Among the four, the most accurate one was the MSHD(8) C, but MSHD(8) D, even though being the second most accurate with an average difference of 2,8% with C, it had considerably fewer cells and a more simple local mesh with just one shell. Therefore, MSHD(8) D was the selected mesh configuration as the number of cells will increase significantly for more complex objects.

Table 6.3 MSHD(8) mesh refinement results with valid simulations highlighted in green (simplified and sorted by ascendant number of mesh cells)

	Mesh cells	CPU Time	Fx (Drag)	Fy (Lift)
MSHD(8) B	4.417.546	3:57:36	1,634	-6,542
MSHD(8) G	6.248.742	26:23:57	1,579	-6,397
MSHD(8) D	6.758.088	6:39:42	1,644	-6,411
MSHD(8) A	9.410.969	12:32:12	1,785	-7,385
MSHD(8) C	9.713.955	9:39:11	1,726	-6,329
MSHD(8) F	12.461.403	112:50:57	1,943	-9,146
MSHD(8) E	13.344.826	36:03:13	1,818	-7,796

For each part studied the mesh configuration will be with the obtained global mesh (global mesh 5: 187 cells in the x direction, 103 in the y direction and 97 in the z direction) (global mesh levels used in Fig. A.1) and a local mesh with just one cell. The width of the local mesh shell will be adapted to each simulation to enclose large gradients. And finally, the computational domain will also be adjusted for every part.

7. FSAE AERODYNAMICS DESIGN

After all the information explained in the previous chapters, it is time to start designing the car. This chapter shows the design of each aerodynamic device **(see Fig. 7.1).** Each element is studied starting from its objective, what rules need to follow, how it is designed and finally tested with simulations.

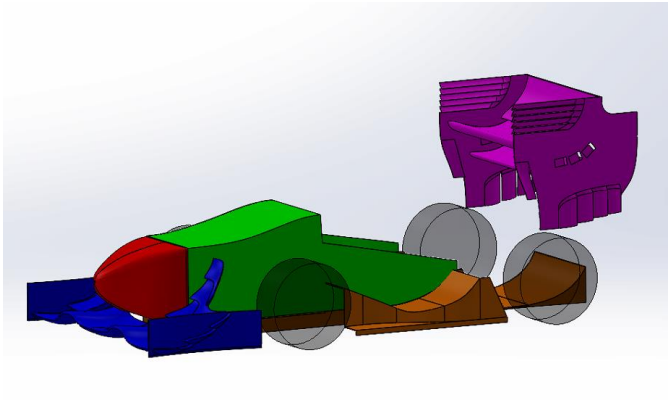

Fig. 7.1 3D design with aerodynamic devices highlighted in different colors

7.1. Nose cone

This section studies the nose cone. It is the simplest aerodynamic device and one of the first elements to impact the air.

7.1.1. Objective

The nose cone is mounted at the front of the car. It aims to deflect the air smoothly to reduce the drag generated. Therefore, a laminar boundary layer with no separation will be preferred.

7.1.2. Aero rules

The Nose Cone has the following security rule regards its radius:

“The bodywork in front of the front wheels must have a **radius of at least 38 mm extending at least 45°** relative to the forward direction, along the top, sides and bottom of all affected edges.” – From Formula Student Rules 2022 (see [12]), T2.3.4, pg. 26

7.1.3. 3d design

The starting point of creating the nose cone is sketching the chassis front part **(see Fig. 7.2a)** as that will be the size of the bodywork that follows the nose cone. In a perpendicular plane, the profile of the nose cone will be drawn **(see Fig. 7.2b)** taking care of not breaking the rule stated before. The profile is moderately asymmetrical as we want more air coming down and entering the undertray. This will generate a bit of upforce, but the increase of air in the undertray should have a greater effect on the downforce and will compensate for this small upforce. Finally, an intermediate plane should be created **(see Fig. 7.2c)** to adjust the horizontal shape before making a surface loft **(see Fig. 7.2d).**

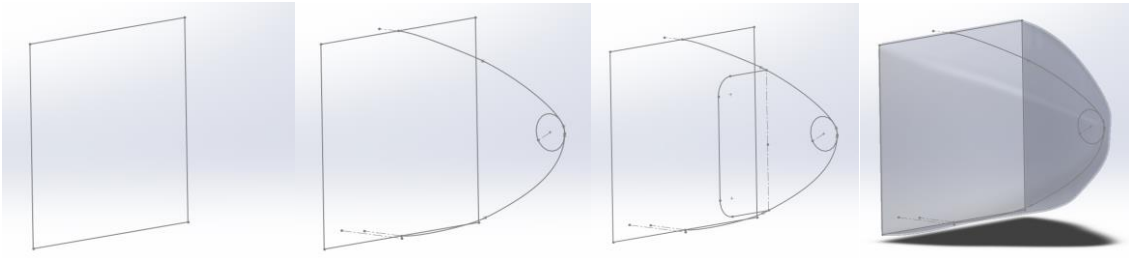

Fig. 7.2 Nose cone development

The nose cone shape can be easily modified by adjusting just two sketches according to the simulations. Moreover, to move more air towards the undertray, the nose cone's lower side can be reshaped with a concave face **(see Fig. 7.3).**

Fig. 7.3 Nose cone with concave lower side

7.1.4. Simulations

The nose cone is located very close to the ground (33 mm for the simulations), therefore a moving ground needs to be added. The ground and air have the same velocity and direction to simulate a car moving through steady air.

After refining the mesh (see Fig. A.2 and Table A.3), the nose cone simulated generates a drag of 5,024 N and a lift of 1,296 N. As stated above, the nose cone aims to reduce drag. Therefore, the results can be considered valid. Lift has a positive value because the nose cone is redirecting air downwards to the undertray. But the lift value is so small that it should be compensated by the undertray.

The asymmetry of the nose cone can be easily identified by looking at the pressure plot **(see Fig. 7.4).**

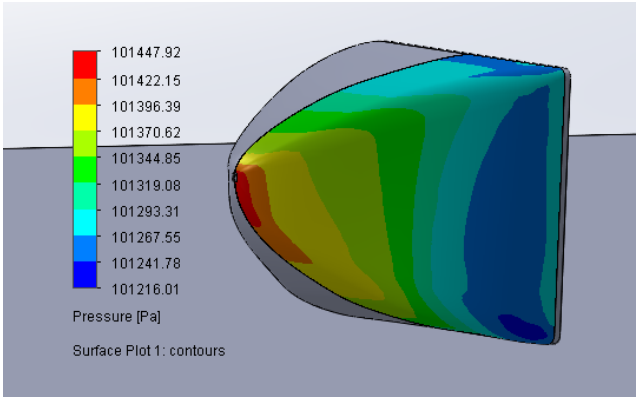

Fig. 7.4 Nose cone pressure plot

7.2. Undertray and sidepots

This section studies the undertray and sidepots. This element follows the nose cone and is located under the car.

7.2.1. Objective

The undertray generates a high amount of downforce using the air below the car. For this purpose, the air gets accelerated by forcing it to go through a narrower area (Bernoulli's principle); this can be done because it is very close to the ground (Ground Effect). Moreover, the air exits the undertray through the diffuser. The diffuser deflects the air upwards to create a downward force (Newton's third law).

On the other hand, the sidepots are responsible for cooling, but they can also be designed to produce a bit of downforce without creating a lot of drag. In this car, they will be attached to the undertray, but they could also be an independent part.

7.2.2. Aero rules

“The minimum static **ground clearance** of any portion of the vehicle, other than the tires, including a driver, **must be 30 mm**.” – From Formula Student Rules 2022 (see [12]), T2.2.1, pg. 25

“Sliding skirts or other aerodynamic devices that by design, fabrication or as a consequence of moving, **contact the track surface are prohibited**.” – From Formula Student Rules 2022 (see [12]), T2.2.2, pg. 25

These rules are more dependent on the suspension of the car but are important to take into account for simulations and to have an idea of how close will be the ground. Moreover, the volume is defined by the following rules:

“No part of the vehicle may enter a **keep-out-zone defined by two lines extending vertically from positions 75 mm in front of and 75 mm behind the outer diameter of the front and rear tires** in the side view of the vehicle, with steering straight ahead. This keep-out zone extends laterally from the outside plane of the wheel/tire to the inboard plane of the wheel/tire assembly.” – From Formula Student Rules 2022 (see [12]), T2.1.3, pg. 25

“All aerodynamic devices forward of a vertical plane through the rearmost portion of the front face of the driver head restraint support, excluding any padding, set to its most rearward position, must be **lower than 500 mm from the ground**.” – From Formula Student Rules 2022 (see [12]), T8.2.1, pg. 52

“All aerodynamic devices lower than 500 mm from the ground and further rearward than the front axle, **must not be wider than a vertical plane touching the most outboard point of the front and rear wheel/tire.**” – From Formula Student Rules 2022 (see [12]), T8.2.2, pg. 52

“All aerodynamic devices **must not extend further rearward than 250 mm from the rearmost part of the rear tires.**” – From Formula Student Rules 2022 (see [12]), T8.2.3, pg. 52

7.2.3. 3d design

Prior to the current family of undertrays studied in this project, lots of alternative designs were made and studied **(see Fig. 7.5)**. They will not be covered here as the current family implements everything that has been learnt by the older ones.

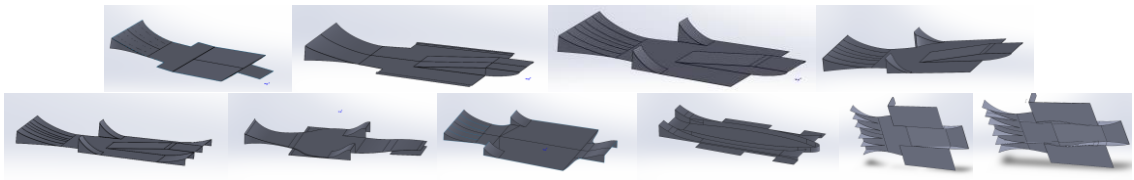

Fig. 7.5 Evolution of undertray designs

The current undertray **(see Fig. 7.6)** incorporates the sidepots and consists of simple and smooth surfaces. The incoming air gets split in two by the undertray to force the air through a narrower area. Then the air exits through the diffuser, which incorporates some guides that reduce mixing and therefore drag. Finally, it is hard to notice in plain sight, but the undertray is a bit tilted (rake) improving its efficiency.

The sidepots act similarly but with a radiator at the end. The air enters through a big area, gets accelerated through a narrower section and finally gets deflected upwards towards the radiators.

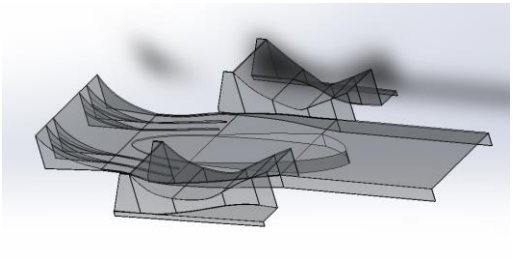

Fig. 7.6 Current design of undertray and sidepots.

7.2.4. Simulations

The undertray is also very close to the ground and the simulations need a moving ground as for the nose cone simulations.

The undertray is simulated without the nose cone or the bodywork and therefore the obtained forces should not be considered (results from refinement in Fig. A.3 and Table A.4). The air flowing through the upper side of the undertray is moving free during the simulation but the undertray will be mounted below the car and no air will flow through the upper side of the undertray.

Therefore, the simulations are used to check if the air detaches the undertray at the diffuser or the sidepots and if it is correctly accelerated. Looking at a pressure plot of the undertray's lower side **(see Fig. 7.7)** it can be seen that the sidepots work perfectly but the undertray is not as effective as the sidepots.

To check if the air is being accelerated just need to check if the pressure around the middle is lower than at the front part. Lower pressure is caused by fast air and will induce a down force.

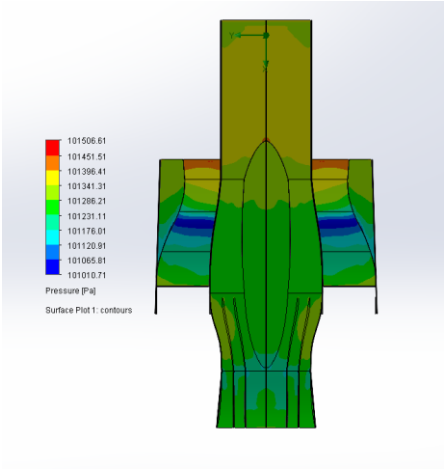

Fig. 7.7 Pressure plot of the undertray lower side

The undertray is not able to generate the desired lift due to rules limitations. The competition rules impose a 30 mm ground clearance and due to the dynamic design of the car, there is a very small room for the undertray and the air cannot be accelerated as it should be.

Even though the undertray does not generate a big amount of force by accelerating air, it can generate downforce by pushing air upwards through the diffuser. Therefore, it is important to ensure the air does not detach the diffuser. This can be checked by looking if there is a low pressure zone at the tip of the diffuser or by illustrating the particle trajectories of the simulation **(see Fig. 7.8).**

The simulation confirms the air is pushed upwards at the cost of generating a turbulent zone that will cause drag.

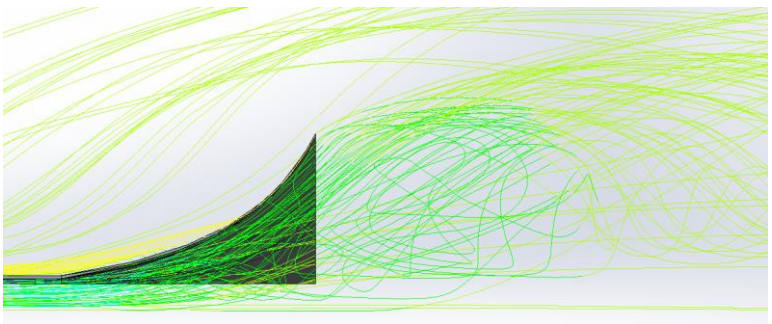

Fig. 7.8 Particle trajectories around the diffuser

7.3. Front wing

This third section is about the front wing. This element is more complex than the previous ones but it can be better related to aviation.

7.3.1. Objective

The front wing is the first aerodynamic device that encounters the free air with the mission of producing downforce. It acts similarly to an airplane wing but generates a force downwards instead of upwards. The front wing is usually composed of multiple elements chained together and behaving like static flaps. This setup generates more downwards force at the expense of increasing drag. Front wings also have some plates at their sides called endplates. They avoid mixing air between the upper and lower side of each wing.

Designing the front wing is not straightforward as, besides creating its force, it defines how the air will encounter the rest of the car. Hence, a well-designed front wing should also redirect air to the other aerodynamic elements of the vehicle.

7.3.2. Aero rules

“All aerodynamic devices in front of the front axle and extending further outboard than the most inboard point of the front tire/wheel **must be lower than 250 mm from the ground.**” – From Formula Student Rules 2022 (see [12]), T8.2.1, pg. 52

“All aerodynamic devices lower than 500 mm from the ground and further rearward than the front axle, **must not be wider than a vertical plane touching the most outboard point of the front and rear wheel/tire**” – From Formula Student Rules 2022 (see [12]), T8.2.2, pg. 52

“All aerodynamic devices **must not extend further forward than 700 mm from the fronts of the front tires**” – From Formula Student Rules 2022 (see [12]), T8.2.3, pg. 52

From the 3 rules above, a 3d volume can be created and the front wing must be contained within the volume. Moreover, as the front wing is the first element to contact with something in case of an accident, it must fulfil the following rule about the edges:

“All forward facing edges of aerodynamic devices that could contact a pedestrian must have a **minimum radius of 5 mm for all horizontal edges and 3 mm for vertical edges**.” – From Formula Student Rules 2022 (see [12])., T8.3.1, pg. 52

7.3.3. 3d design

Like the undertray, the front wing is a very complex part, and it is hard to select the path to follow **(see Fig. 7.9)**. The front wing is made of two distinct elements: the wings and the endplates. Unlike the rear wing, the front wing has very limited available volume and the endplates tend to be simple shapes that just avoid the mixing of air between the upper and lower part of the wing.

The first thing to define is which airfoil should be used and then adjust the form and the angle of the wings.

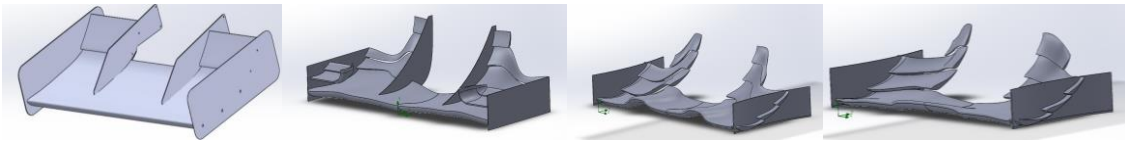

Fig. 7.9 Evolution of front wing designs

A high lift airfoil will be chosen for the wings as the drag at the car's average velocities (between 50 and 100 km/h) is not predominant. Typical high lift airfoils are FX74_CL5_140, S1223, LNV109A **(see Fig. 7.10)**.

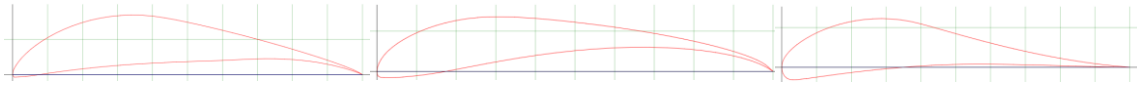

Fig. 7.10 FX74_CL5_140 (Left), S1223 (Center), LNV109A (Right) [26]

The best thing to do would be to test a few airfoils using 2D simulations, choose the best airfoil for the current problem and optimize it in 2D and 3D. This process would take thousands of simulations and this project focuses on the overall aerodynamics of the car. Therefore, all of the airfoils above are valid and the chosen one is the S1223 because it will be the easiest one to build and chain.

Finally, the selected airfoil is used to create all the wings of the front wing. Again, the best way to shape the front wing would be by doing 3d simulations and optimizing the angle of attack, the scale and the separation for all the wings. These are a lot of variables and would require a big amount of CPU time and a large team to do it. But this is not the objective of the project. Instead, the designs are made based on aerodynamic theory and intuition to later compare them in simulations.

7.3.4. Simulations

After refining the mesh (see Fig. A.4 and Table A.5), the second step is to test the positioning of the wings and check if there is enough gap between them to let the flow pass without slowing it down. From the simulations **(see Fig. 7.11),** the flow is working as expected for the first three wings. The pressure is lower at the bottom of the wing and higher at the top. Furthermore, the air is being pushed upwards following the wings structure.
But the fourth wing, the smallest one, seems to be experiencing some air detachment. A surface pressure plot helps to check if that is the case. But from the simulations **(see Fig. 7.12)** it seems that there is no abnormal low pressure area at the fourth element and the air follows its shape until it reaches the trailing edge.

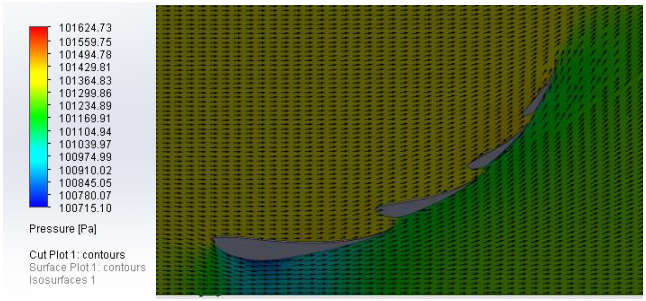

Fig. 7.11 Section cut of a front wing pressure plot

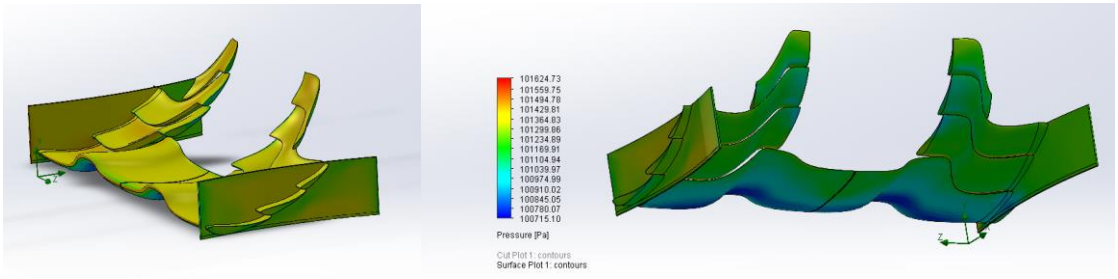

Fig. 7.12 Surface pressure plot of the front wing. Top view (Left) and bottom view (Right)

On the other hand, the simulations' numerical outputs seem a bit lower than the expected values. The total Lift is -27,144 N and the total Drag is 16,168 N. Even though the lift to drag ratio is quite good (-1,67), the values are significantly smaller than the ones of the rear wing (explained in the following section).

To check if the numbers are correct and the front wing can be considered valid, let's analyze the particle study **(see Fig. 7.13).** From it, it can be noticed that the lateral sides of the front wing, which are shorter to ensure enough air enters the sidepots, divert significantly less air that the center part. Moreover, at the center,

the air is being mixed from the upper side to the lower pressure down side generating a vortex.

In a nutshell, the front wing can be considered valid because there is not much room to add bigger wings and the vortices, that at first generate drag, will help the air attach to the bodywork due to its turbulent characteristic. In addition, it is normal to have higher values for the rear than for the front wing in low speed formula cars because of the volume comparison of both devices.

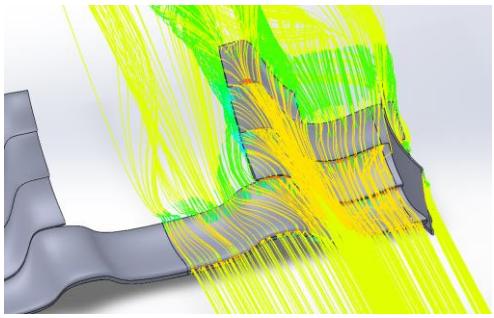

Fig. 7.13 Particle simulation for the front wing

7.4. Rear wing

This section is about the rear wing. It is a similar device to the previous front wing but because it is situated behind the car it needs a bit more thought when designing it.

7.4.1. Objective

The rear wing has the same objective as the front wing, to generate a down force. The main difference is that the rear wing is located at the back of the car. Hence, the incoming air is turbulent, and there is no need to worry about how the air exits the rear wing. Because of this, the rear wing is usually larger than the front wing and has very complex endplates to maximize efficiency.

Moreover, being the last element of the car allow it to be modified without affecting the other devices' behavior. This advantage is why the rear wing can be designed with two possible configurations, one to maximize down force and take turns faster (convenient for Skidpad, Autocross, Endurance and efficiency events) and another one to minimize drag (useful for Acceleration event).

7.4.2. Aero rules

“All aerodynamic devices rearward of a vertical plane through the rearmost portion of the front face of the driver head restraint support, excluding any padding, set to its most rearward position **must be lower than 1.2 m from the ground.”** – From Formula Student Rules 2022 (see [12]), T8.2.1, pg. 52

“All aerodynamic devices higher than 500 mm from the ground, **must not extend outboard of the most inboard point of the rear wheel/tire**.” – From Formula Student Rules 2022 (see [12]), T8.2.2, pg. 52

“All aerodynamic devices must not extend further rearward than **250 mm from the rearmost part of the rear tires**.” – From Formula Student Rules 2022 (see [12]), T8.2.3, pg. 52

From the 3 rules above, a 3d volume can be created and the rear wing must be contained inside it.

7.4.3. 3d design

The rear wing will be composed of three elements: the main wings, the beam wing and the endplates. At the back of the vehicle, there will be three elements displacing air upwards with similar flow directions to increase efficiency **(see Fig. 7.14).** These 3 elements are the main wings of the rear wing (composed of two wings), the beam wing (one wing situated below the main wings that allow connecting the diffuser and the main wings flow) and the diffuser (placed at the end of the undertray).

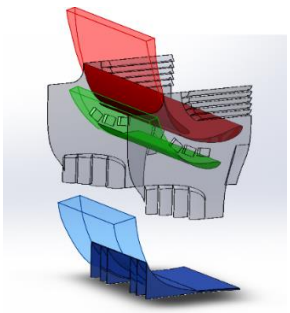

Fig. 7.14 Diffuser and rear wing flow directions

As for the front wing, a high lift airfoil will be chosen for the wings. In addition to the previous airfoils **(see Fig. 7.10),** a new airfoil called MSHD **(see Fig. 7.15)** should be considered.

MSHD will be the selected one as it is a special airfoil designed specifically for this competition by Pakkam, Sriram Saranathy (see [1]).

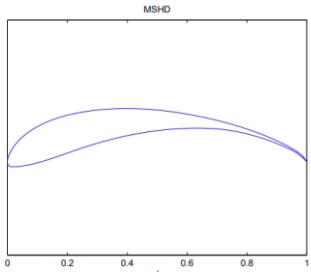

Fig. 7.15 MSHD airfoil profile [1]

The endplate is the last element of the rear wing. It aims to increase the efficiency of the main and beam wings by avoiding undesired air mixing. For this purpose, it can be made of different elements that work with the pressure difference between the fast-moving air between the endplates and the calm air outside. The following endplate elements will be studied:

7.4.3.1. Endplate slats

They are located at the rear bottom part of the endplates. The slats are extensions twisted outwards **(see Fig. 7.16)** that aim to expand the airflow that exits the diffuser, increasing the efficiency of the diffuser. The low pressure air from under the beam wing is deflected by the slats and pulls the diffuser's airflow outwards **(see Fig. 7.17).**

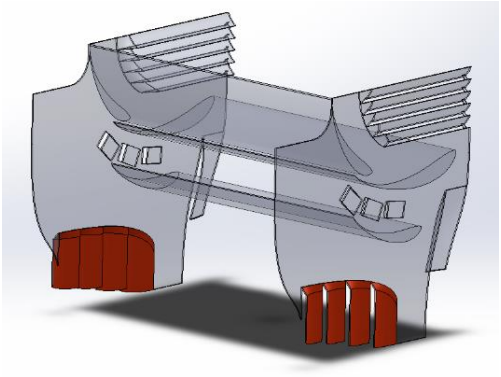

Fig. 7.16 Endplate slats

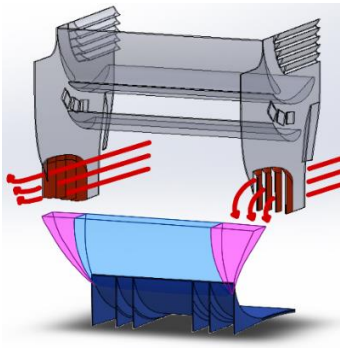

Fig. 7.17 Effect of the slats on the diffuser's flow

7.4.3.2. Cutout

The top rear part of the endplate is usually cut **(see Fig. 7.18)** to allow the neutral air outside the endplates to mix with the low pressure accelerated one from below the main wings **(see Fig. 7.19).** Theoretically, this doesn't reduce the pressure gradient but adds kinetic energy to it, strengthening the underwing vortex.

An additional side effect of the cutout is the generation of extra small vortices that can counter the existing ones if they rotate in an opposite direction or reinforce the existing ones if they are co-rotating **(see Fig. 7.20).**

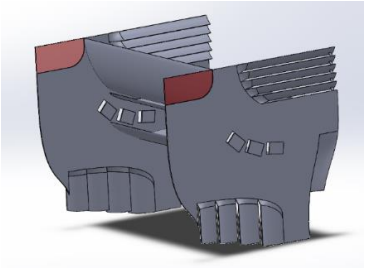

Fig. 7.18 Endplate cutouts

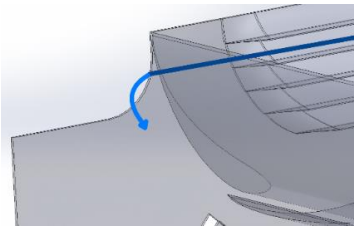

Fig. 7.19 Cutout main effect

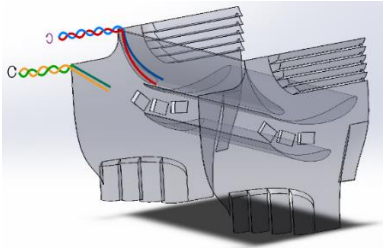

Fig. 7.20 Cutout side effects

7.4.3.3. *Endplate louvres*

They are horizontal cuts at the top front side of the endplate **(see Fig. 7.21).** They allow the high-pressure air above the main wings to flow towards the neutral air outside the endplate **(see Fig. 7.22).** Hence, minimizes a little bit the induced drag caused by the wingtip vortex but also reduces the downforce in a small area of the wing close to the endplates.

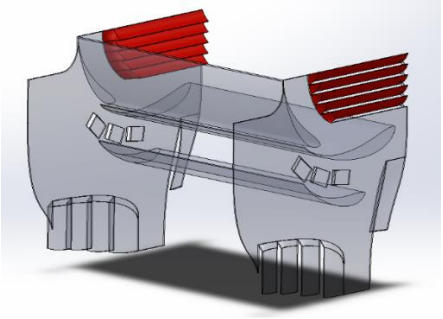

Fig. 7.21 Endplate louvres

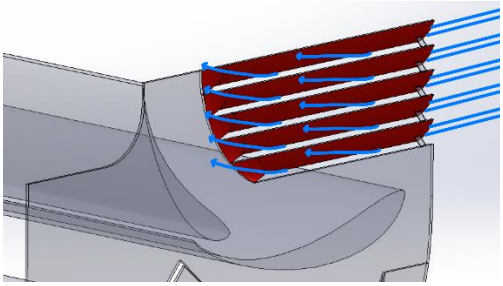

Fig. 7.22 Airflow through endplate louvres

7.4.3.4. Slots

They are located below the main wing **(see Fig. 7.23).** They work similarly to the louvres, but as the air below the main wing has lower pressure than the air outside the endplate, this is pulled inwards instead of outwards **(see Fig. 7.24)**. The effect is theoretically the same, the induced drag and downforce are reduced.

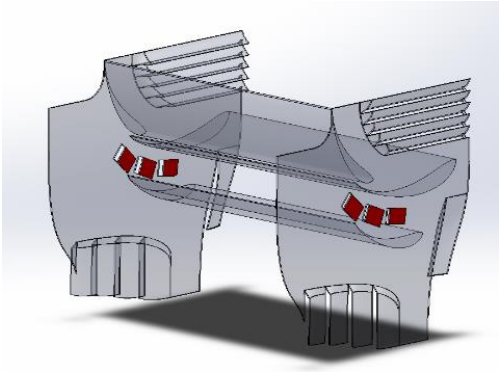

Fig. 7.23 Endplate slots

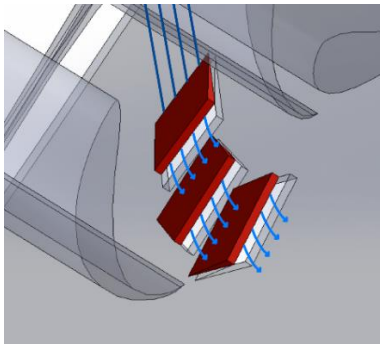

Fig. 7.24 Airflow through endplate slots

7.4.3.5. Endplate leading edge slot

The lower front part of the endplate is usually shaped in a way that tries to speed up the flow below the main wing **(see Fig. 7.25).** This air is usually very turbulent and can be negative to the rear wing efficiency if not done correctly.

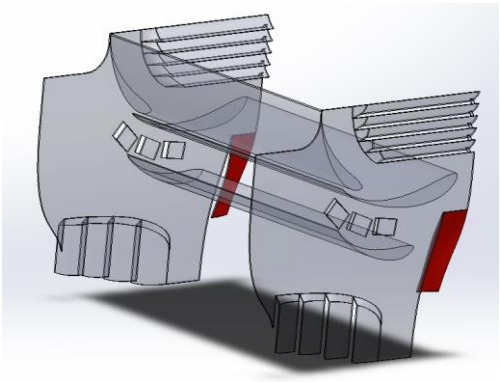

Fig. 7.25 Endplate leading edge slot

7.4.4. Simulations

For the rear wing, every component of the endplates explained above needs to be tested to know if they are somewhat useful for the car. Therefore, 4 different versions were created by adding new elements to each version **(see Fig. 7.26).**

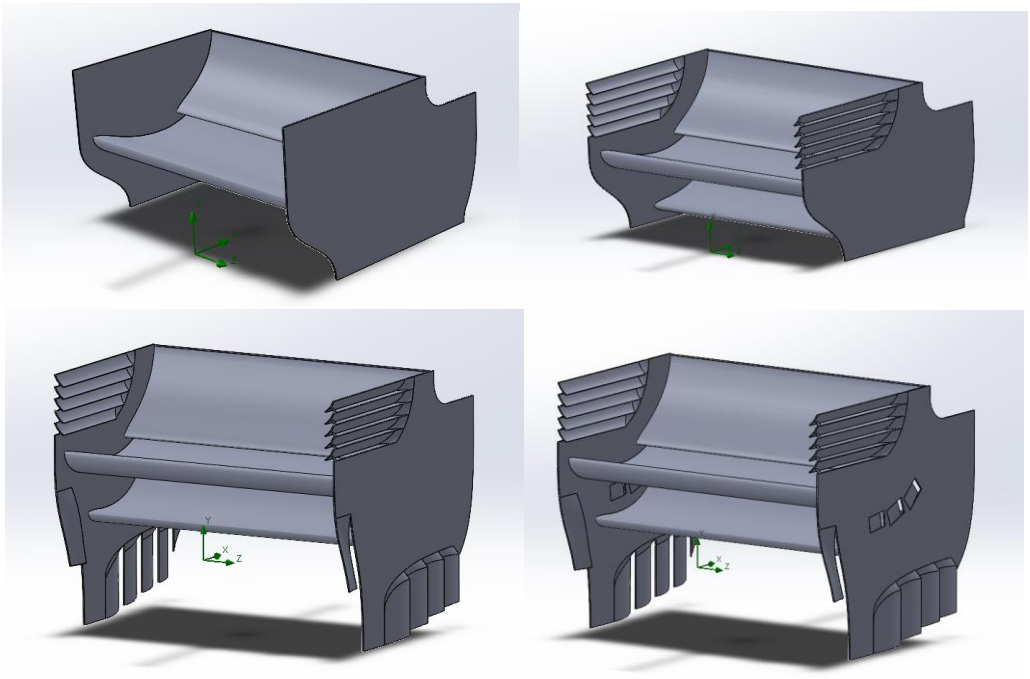

Fig. 7.26 Rearwing v1 (Top-Left), v2 (Top-Right), v3 (Bottom-Left) and v4 (Bottom-Right)

After adjusting the mesh for each version (see Fig. A.5, Fig. A.6, Fig. A.7, Fig. A.8, Table A.6, Table A.7, Table A.8 and Table A.9), the results of the simulations can be summarized in the following table **(see Table 7.1):**

Table 7.1 Simulation results for rear wings v1, v2, v3 and v4

	Drag (Fx)	**Lift (Fy)**	**-L/D (-Fy/Fx)**
Rearwing v1	22,578 N	-64,971 N	2,878
Rearwing v2	30,443 N	-68,812 N	2,260
Rearwing v3	28,671 N	-64,192 N	2,239
Rearwing v4	56,012 N	-122,925 N	2,195

Lift to drag ratio decreases with each version but at the same time lift is significantly higher with each version too. Therefore, for a straight circuit where lift has less importance, v1 would be the chosen version. Ideally, two separate endplates should be made so the team can choose from the high lift or high lift to drag ratio depending on the event.

The competition do not allow to change between different devices during the competition so the selected version will be v4 because of the close turns in Autocross and Skidpad events.

Once the rear wing version is selected, let's check if the air is behaving as expected. A pressure plot **(see Fig. 7.27)** helps to identify how the air close to the surface is behaving. High pressure air (red color) is equivalent to slow air and low pressure air (blue color) is equivalent to fast moving air. The designed rear wing has warmer colors on the upper side of each wing which indicates they are working correctly and will generate lift.

Moreover, the trailing edge of the main and beam wing does not have a lower pressure area, meaning there is no undesired detachment.

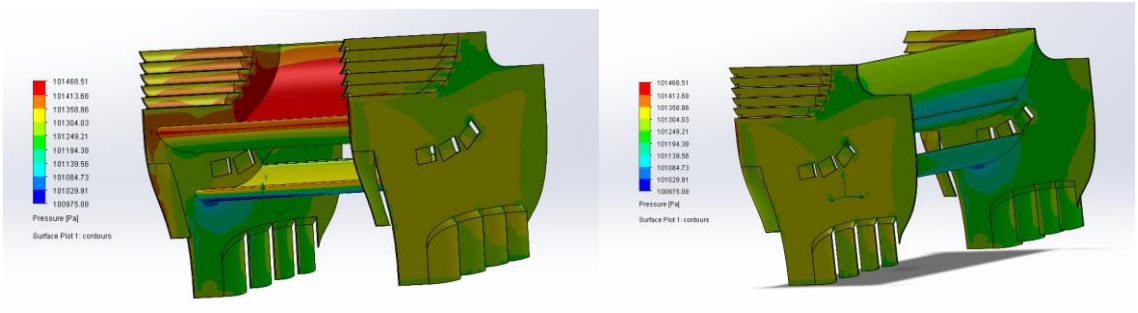

Fig. 7.27 Rear wing surface pressure plot. Front view (Left) and rear view (Right)

A flow trajectory plot **(see Fig. 7.28)** is the easiest way to visualize how the air is being pushed upwards (upwash) generating an opposite force downwards (Newton's third law).

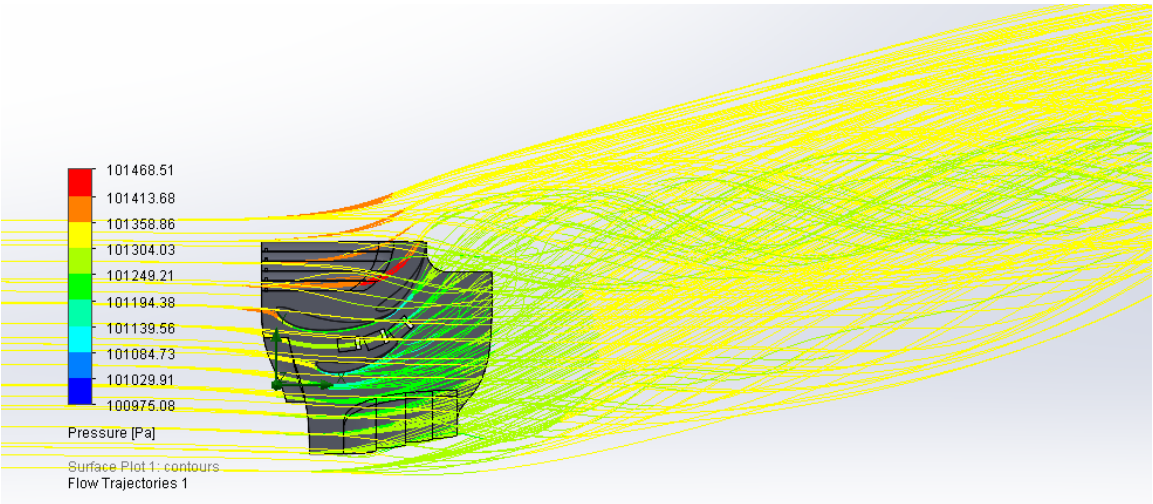

Fig. 7.28 Undertray flow trajectory plot (side view)

7.5. Bodywork

This section explains the design of the bodywork. The bodywork can be understood as the aerodynamic cover of the chassis. This section is a bit different to the previous ones as there are no simulations because the aerodynamic impact of the bodywork can only be appreciated in the full car simulations.

7.5.1. Objective

The bodywork's objective is to cover the chassis to avoid air flowing inside the car and improve aerodynamic efficiency.

7.5.2. Aero rules

The bodywork must comply with all the rules specified above about the available volumes and it must also comply with the following rule:

"There must be **no openings through the bodywork into the cockpit other than that required for the cockpit opening**. Minimal openings around the front suspension and steering system components are allowed." – From Formula Student Rules 2022 (see [12]), T2.3.1, pg. 26

7.5.3. 3d design

The simplest way to cover the chassis would be to use cut plates for each plane of the chassis **(see Fig. 7.29).** This method would require a significant amount of metal cutting and welding. Moreover, its sharp edges **(see Fig. 7.30)** would make the air detach from the surface and generate undesired drag.

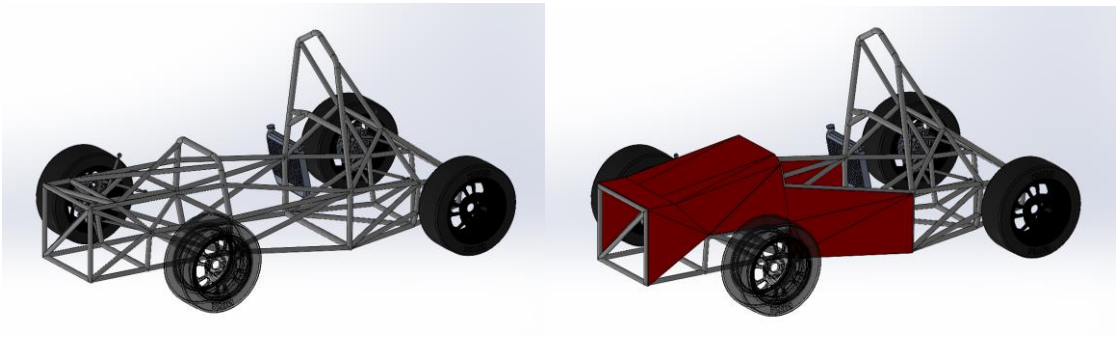

Fig. 7.29 Car chassis (Left) and car chassis with simple bodywork (Right)

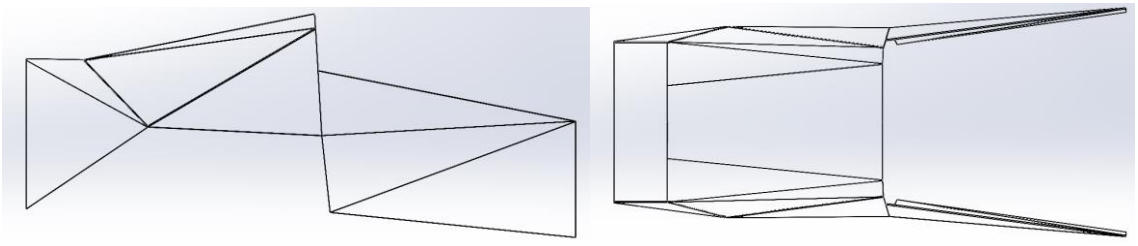

Fig. 7.30 Simple bodywork side view (Left) and top view (Right)

Instead, a bodywork can be designed by using only three plates with a more aerodynamic shape **(see Fig. 7.31 and Fig. 7.32)**. This design covers the chassis using smoother surfaces to reduce drag.

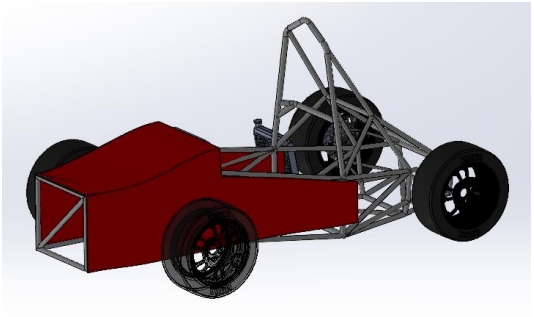

Fig. 7.31 Car chassis with smooth bodywork

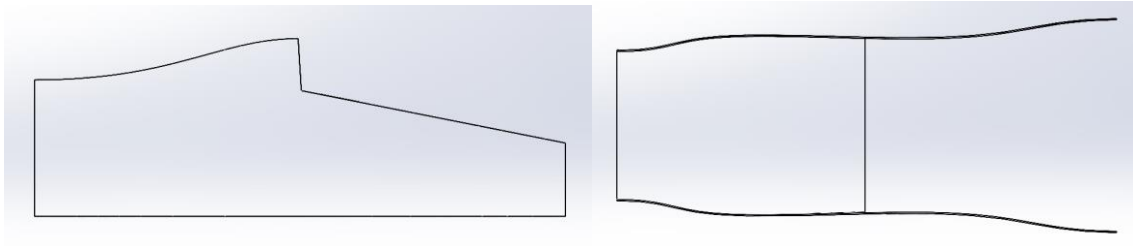

Fig. 7.32 Smooth bodywork side view (Left) and top view (Right)

7.6. Final result

In this final section, all the aerodynamic devices will be assembled to study the whole car aerodynamics.

7.6.1. Simulations

For this last simulation, all the designed devices are assembled as they will be mounted on the car with the ground placed at 35 mm of the undertray. Creating a simulation will all the devices would be very CPU consuming. Therefore, the full car simulation is composed only of the aerodynamic devices and the wheels **(see Fig. 7.33).**

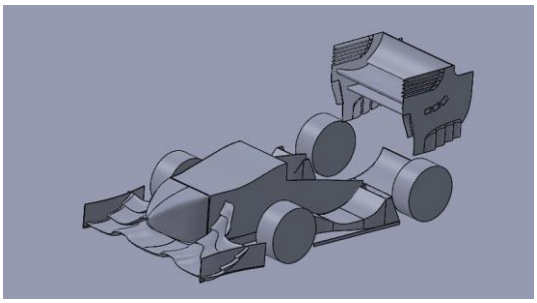

Fig. 7.33 Full car assembly

After the mesh refinement (see Fig. A.9 and Table A.10), the simulation resulted in a total lift of -159,639 N and drag of 118,009 N with a lift to drag ratio of -1,35. The ratio is lower than the individual front and rear one due to the additional drag the bodywork and the wheels generate. Still, the ratio is inside the scope of the project (just as a reference, a real f1 car has a lift to drag ratio of 2,5 on average).

The surface pressure plots **(see Fig. 7.34)** show that the elements generating more downforce are the front wing and rear wing, as they have the higher pressure difference (blue –low pressure- at the bottom part and yellow –high pressure- at the upper part).

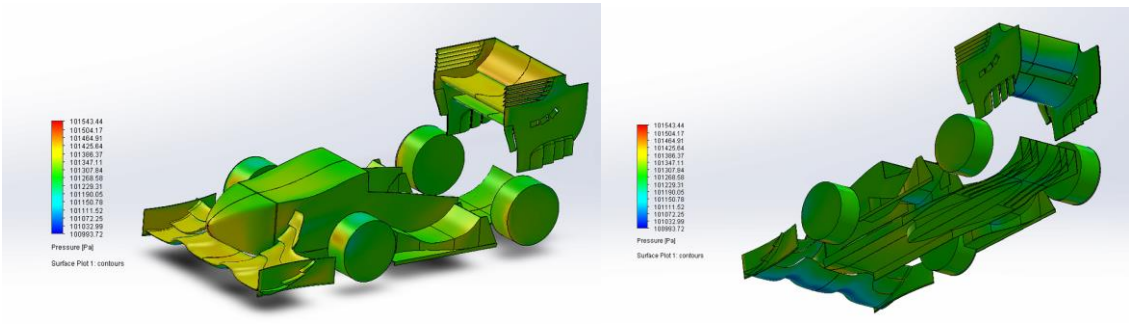

Fig. 7.34 Full car surface pressure plot. Top view (Left) and bottom view (Right)

In the particle simulation **(see Fig. 7.35)** it can be seen how the air is being pushed upward (generating downforce by Newton's third law) and how turbulent zones are being created at the cockpit and diffuser (generating drag as explained in Drag section).

Zooming into the diffuser in the particle simulation **(see Fig. 7.36),** it can be appreciated that the air is not being pushed upwards by the diffuser because it is detaching the surface. This is because of the limitations on the undertray height caused by the chassis and suspension design. Still, the diffuser will work better when the suspension compresses at high speeds and the ground effect gains importance.

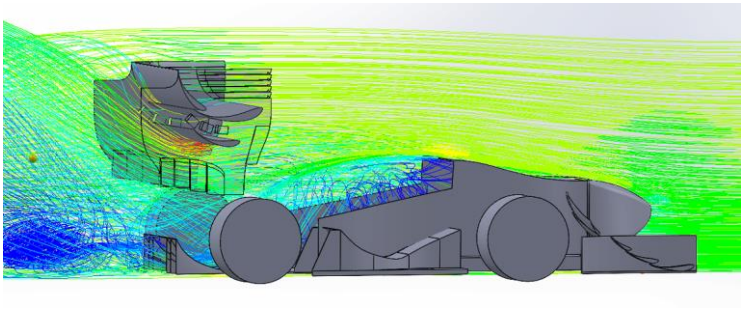

Fig. 7.35 Full car particle simulation

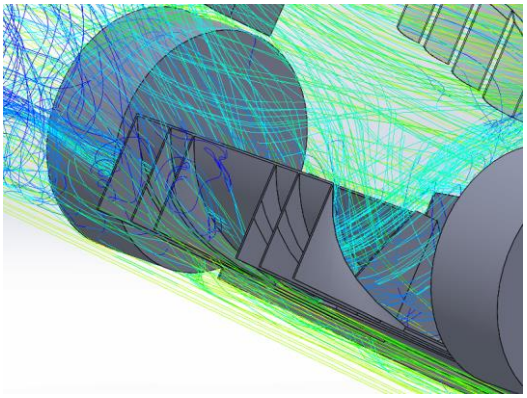

Fig. 7.36 Full car particle simulation (close up to the diffuser)

The aerodynamics devices are not perfect but have a positive impact on the car aerodynamics. Even formula 1 teams, with hundreds of workers and budgets of hundreds of millions do not reach perfection and this is the beauty of aerodynamics, there is always room to improve.

In the following chapter possible future improvements are discussed for next year season design. Some of the methods are frequently used in car aerodynamics and others are more revolutionary ideas that may need some testing to develop.

8. AERODYNAMIC IMPROVEMENTS

As stated in the conclusions of the previous chapter, more simulation time would result in an improvement of the aerodynamic performance. Iterations of simulations can be done to optimize the angle, shape, positioning and size of the wings. Moreover, for some events such as Skidpad, where the car is constantly turning, it would be interesting to improve the aerodynamics of the car when the air is not coming frontally but with a side angle.

But this chapter focuses on additional techniques that would help to improve the efficiency of the car aerodynamics.

8.1. DRS

DRS or Drag Reduction System in f1 consists of a movable flap in the rear wing. By changing the angle of attack, the aerodynamics of it also vary **(see Fig. 8.1)**, allowing the car to have the necessary lift in corners and reduce the unnecessary drag in straights.

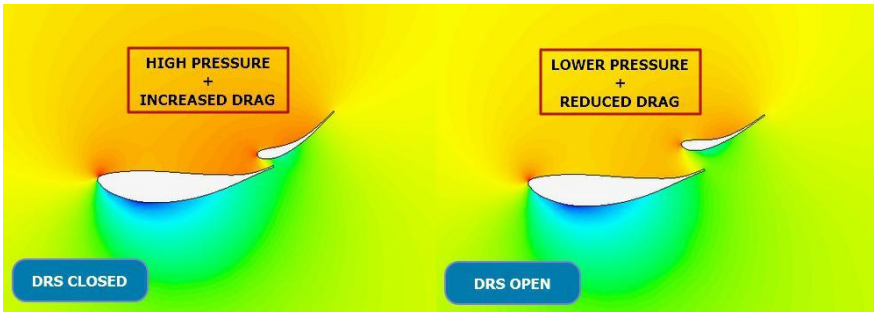

Fig. 8.1 DRS closed and open comparison [27]

This system can also be applied in formula SAE and other team studies have achieved a reduction of almost four times less drag in the open position (see [28]). And other teams have pushed the idea forward and have also found the best angle of attack for each flap and the perfect time to open it (see [29]).

DRS provide will provide a big performance boost on Acceleration events, but there is a more complex way that automatically changes the aerodynamic performance of the wings without the need for an actuator. These are commonly called flexi-wings.

8.2. Flexi-wings

This method takes advantage of the natural bending of the airfoil. When the car is travelling at high speeds the force exerted on the airfoil is also higher and the wing tends to deform elastically. By studying the materials of the wing, this bend can be controlled and be used as an automatic DRS and will be useful for all the events.

This technique can be used for the rear and front wings. There is a research from the American University of the Middle East about this topic (see [30]).

There is also another kind of flexible wing used in f1 until its prohibition. This consist of making the whole rear wing tilt and therefore changing the angle of attack **(see Fig. 8.2).**

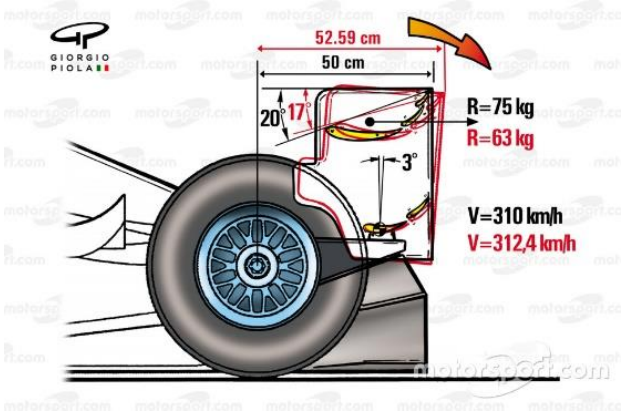

Fig. 8.2 Example of a flexible wing tilt [31]

8.3. Inside ducts

This is one of the solutions that came up in a reunion with the chassis team to solve the shortage of space for the undertray. The idea is to create ducts that move air from the undertray upwards through the chassis with the idea of accelerating the air flowing under the car. This idea was discarded because of schedule limitations but it could be a good idea to develop in the future.

There is also a real case that used this idea. It was from a formula team in 1979 that created a car named Ligier JS11. The team secretly created a hidden duct in the sidepots that would open at a certain pressure. This reduced downforce and drag at high speeds. There is no official documentation because the team intended to hide the technique, but a reporter called Giorgio Piola discovered it during an accident where the car crashed. Then he draws an sketch of the system **(see Fig. 8.3).**

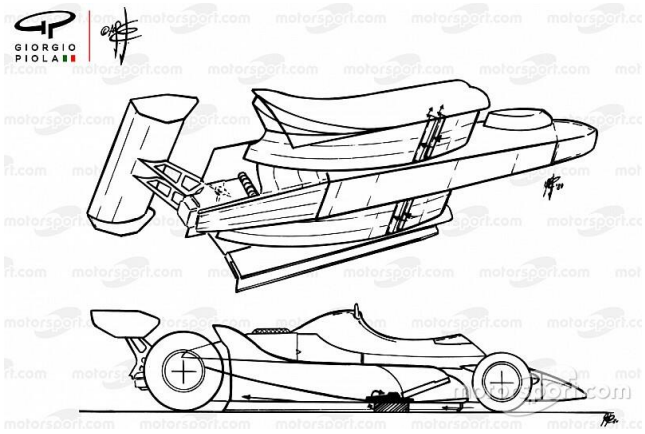

Fig. 8.3 Giorgio Piola sketch of the JS11 hidden flaps [32]

8.4. Gurney flap

The last three methods are not used for generating lift or drag; they are used to avoid the air detaching other aerodynamic devices by making the boundary layer turbulent.

The gurney flap is a small tab at the trailing edge of an airfoil at a 90 degrees angle **(see Fig. 8.4).** It is situated at the higher pressure side with the aim of increasing, even more, its pressure and making the lower pressure air attach to the airfoil.

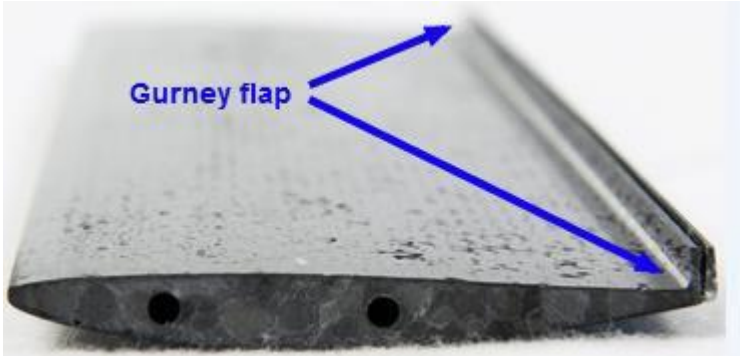

Fig. 8.4 Gurney flap [33]

8.5. Variable surface roughness

This second method consists in creating a rough surface similar to the dimples on a golf ball **(see Fig. 8.5)** (see [34]). The rough surface creates the turbulent boundary layer. This could be used in the diffuser of the designed car to make

the air attach to the surface. The advantage of this method is that it does not need any volume to apply it.

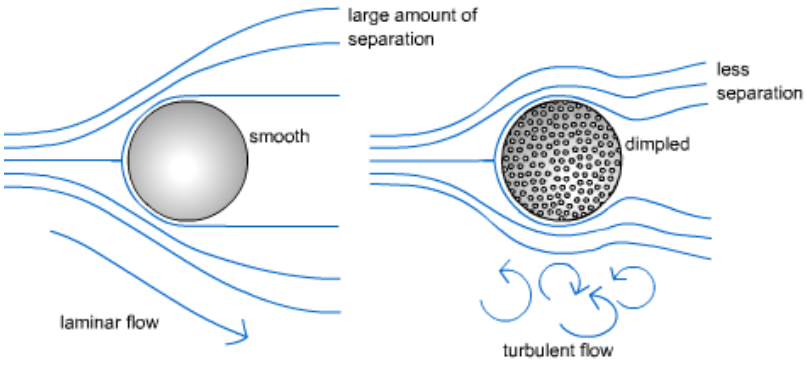

Fig. 8.5 golf ball aerodynamics (right) compared to a smooth ball (left) [35]

8.6. Vortex generators

Remember the purpose of the rear and front wing endplate was to avoid the air mixing between the upper and lower surfaces of a wing. This air mixing results in a wingtip vortex. Vortex generators create this vortex on purpose in order to create a turbulent boundary layer. They could be used along the bodywork of the car or just in front of the tires to reduce their drag generation.

A simple vortex generator consists of a flat plate, but a study by the Technical University of Denmark (see [36]) demonstrated that an aerodynamically shaped vortex generator can improve the lift to drag ratio.

There are infinitely more possibilities to try and with time, resources and knowledge aerodynamics can always be improved.

9. Self-Driving Car

After completing the aerodynamic design, it is time to start with the lap time simulator. This chapter along with the following one will explain the development of the simulator to, later on, test the aerodynamics in the last chapter.

This chapter explains how the program is coded to make a car able to learn how to drive by itself. The idea of this chapter is to understand how the program works without the need of reading its code.

9.1. Implementation

This first section explains the 3 main functions of the program. The simulator is divided in 3 main files: The Car controller, the Neural Network and the Genetic Manager. How the files are correlated will be explained in the last subsection of this section but first, the main functions of each file are explained.

9.1.1. Car physics

The Car controller is responsible of the car's movement by using two input variables: inputPower and inputDirection. These inputs need to be translated into movement using the aerodynamic parameters.

Note that during this subsection the car axis will be with the x axis pointing to the car's moving direction, the y axis pointing to the left side of the car and the z axis pointing opposite to the gravity force **(see Fig. 9.1).**

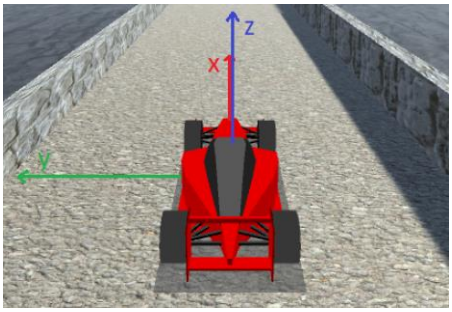

Fig. 9.1 Car axis for car physics

First, acceleration is calculated from the motor force and the drag force. Motor force depend on its given power ($P * inputPower * 1000$) and current velocity (v).

Drag is calculated extrapolating the CFD simulated drag to the actual velocity and adding tire friction.

$$a = \frac{\frac{P * inputPower * 1000}{V} - Drag}{m} \quad \textbf{(9.1)}$$

$$Drag = \frac{V^2}{{V_{ref}}^2} * D_{ref} * 1000 + C_r * m * g + \frac{m * V^2 \tan \alpha_s}{TR} \quad \textbf{(9.2)}$$

$$\alpha_s = maxSlipAngle * inputDirection \quad \textbf{(9.3)}$$

$$TR = \frac{Wheelbase}{\tan \alpha_t} \quad \textbf{(9.4)}$$

Where:

P = Motor's maximum Power [Kw]
$inputPower$ = value from 0 to 1 to accelerate
V = current car's velocity [m/s]
m = car's mass [kg]
V_{ref} = velocity of CFD simulations [m/s]
D_{ref} = drag result of CFD simulations [kN]
C_r = rolling coefficient
g = gravity [m/s2]
α_s = slip angle proportional to the $inputDirection$ [º]
$inputDirection$ = value from 0 to 1 to change direction
TR = Turning Radius [m]
α_t = angle between the wheels' direction and forward direction [º]
$Wheelbase$ = distance between front tires and rear tires [m]

When braking the acceleration is negative and calculated using the vertical force and friction when braking, plus the drag.

$$braking = \frac{\left(\frac{V^2}{{V_{ref}}^2} * L_{ref} * 1000 + m * g\right) \mu_{tire} * inputPower + Drag}{m} \quad \textbf{(9.5)}$$

Where:

L_{ref} = lift result of cfd simulations [kN]
μ_{tire}= friction coefficient when braking

Therefore, in a discrete world, velocity is calculated by increments over small divisions of time. In this case, the program runs over a FixedUpdate, which means 50 times per seconds independent of the frame rate.

$$V_f = V_i + a * \Delta t \quad \textbf{(9.6)}$$

Where:

V_f = next velocity [m/s]
V_i = current velocity [m/s]
a = acceleration or braking value calculated before [m/s2]
Δt = 1/50 [s]

The previous equations allow the car to move realistically in a straight line, but to turn the car, the turn radius at the current velocity needs to be translated into a rotation for every FixedUpdate (α_d). The relation between TR and α_d can be obtained using simple trigonometry.

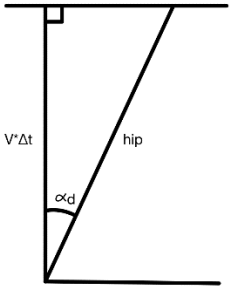

Fig. 9.2 Car's displacement over a FixedUpdate() cycle

From **Fig. 9.2**:

$$\cos \alpha_d = \frac{V*\Delta t}{hip} \rightarrow hip = \frac{V*\Delta t}{\cos \alpha_d} \quad \textbf{(9.7)}$$

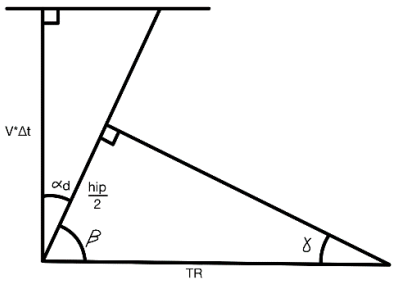

Fig. 9.3 Car's displacement over a FixedUpdate() cycle with Turn Radius (TR)

From **Fig. 9.3**:

$$\beta = 90° - \alpha_d \quad \textbf{(9.8)}$$

$$\gamma + \beta + 90° = 180° \quad \textbf{(9.9)}$$

Applying **eq. 9.8** into **eq. 9.9**:

$$\gamma = 180° - \beta - 90° = 180° - 90° + \alpha_d - 90° = \alpha_d \quad \textbf{(9.10)}$$

Finally applying Pythagoras:

$$\sin\gamma = \frac{hip/_2}{TR} = \sin\alpha_d \quad \textbf{(9.11)}$$

$$TR = \frac{hip/_2}{\sin\alpha_d} = \frac{hip}{2*\sin\alpha_d} \quad \textbf{(9.12)}$$

Using **eq. 9.7**:

$$TR = \frac{V*\Delta t}{2*\sin\alpha_d*\cos\alpha_d} \quad \textbf{(9.13)}$$

And assuming that α_d<< 1:

$$\alpha_d = \frac{V*\Delta t}{2*TR} \quad \textbf{(9.14)}$$

Finally, the ability to skid and lose control needs to be implemented to add more realism to the simulation and don't allow impossible turns at high velocities. High downforce cars are not designed to drift. Therefore, skidding can be applied to the simulation by imposing a skid condition that invalidates the run and restart it.

When the car is turning, centripetal forces try to push the car outwards and tire frictions are responsible to counter these forces. When centripetal forces are higher than the tires lateral friction, the car loses control.

The centripetal force (Fy_{req}) can be easily calculated with the following equation:

$$Fy_{req} = \frac{m*V^2}{TR} \quad \textbf{(9.15)}$$

On the other hand, the lateral force exerted by the tires depend on the vertical force exerted on each tire and the grip equation of the tire. The equation to obtain μ for this case is an empirical equation obtained experimentally by a team member.

$$Fy = 2 * Fy_{out} + 2 * Fy_{in} \quad \textbf{(9.16)}$$

$$Fy_{out/in} = Fz_{out/in} * \mu_{out/in} \quad \textbf{(9.17)}$$

$$\mu_{out/in} = 1{,}82 - \frac{\left(\frac{Fz_{out/in}}{1000}\right)^2}{45} \quad \textbf{(9.18)}$$

$$Fz_{out} = Fz_{aero} + Fz_{static} + Fz_{LoadTransfer} \quad \textbf{(9.19)}$$

$$Fz_{in} = Fz_{aero} + Fz_{static} - Fz_{LoadTransfer} \quad \textbf{(9.20)}$$

$$Fz_{LoadTransfer} = \frac{m*h_{cg}*V^2}{2*track*TR} \quad \textbf{(9.21)}$$

$$Fz_{static} = \frac{m*g}{4} \quad \textbf{(9.22)}$$

$$Fz_{aero} = \frac{\frac{V^2}{V_{ref}^2}*L_{ref}*1000}{4} \quad \textbf{(9.23)}$$

Where:

Fy_{out} = Lateral force exerted on an outside tire [N]
Fy_{in} = Lateral force exerted on an inside tire [N]
Fz_{out} = Vertical force exerted on an outside tire [N]
Fz_{in} = Vertical force exerted on an inside tire [N]
Fz_{aero}=Vertical force on a tire due to aerodynamics [N]
Fz_{static}= Vertical force on a tire due to its weight [N]
$Fz_{LoadTransfer}$= difference between vertical loads on the inside and outside tires
h_{cg}= car's center of gravity height [m]
$track$=distance between outside and inside tires [m]

This set of equations will make the car move more realistically to obtain more accurate results.

9.1.2. Neural network function

The neural network file is the one responsible of creating the networks. The neural network is created using two variables: the number of hidden layers and the number of hidden neurons. Every hidden layer will have the same number of hidden neurons **(see Fig. 9.4).**

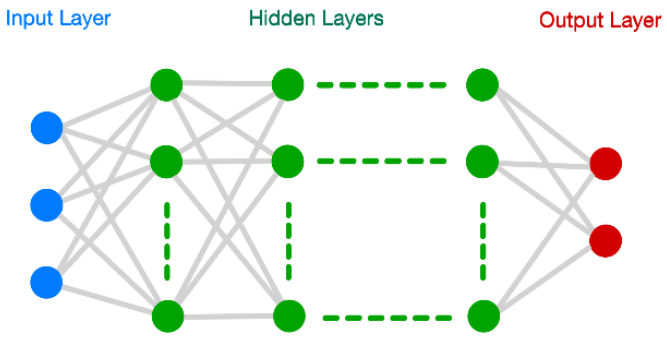

Fig. 9.4 Neural network with variable number of hidden neurons and layers

Each neural network is represented by a list of weight matrices. The first matrix contains the weights of all the connections from the inputs to the first hidden layer. Hence, this matrix has three rows (number of inputs) and the same number of columns as the number of hidden neurons. Thus, the first hidden layer can be obtained by multiplying the input matrix and the first weigh matrix **(see Fig. 9.5).**

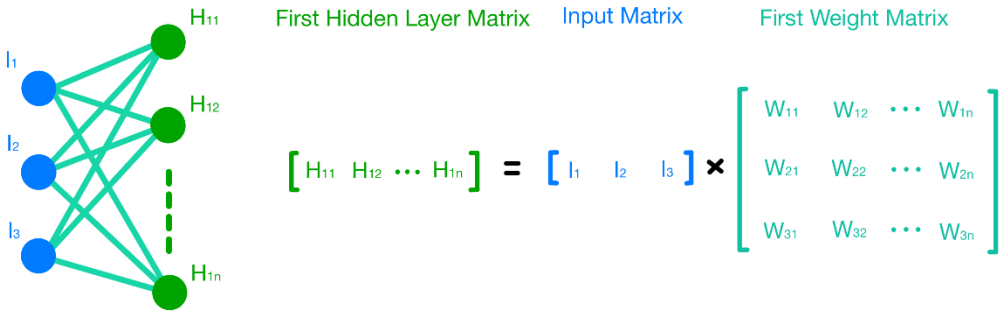

Fig. 9.5 Input to first hidden layer matrix multiplication

The following are square matrices with a size equal to the number of hidden neurons of each hidden layer. These contain the weigh values of the connections between hidden layers **(see Fig. 9.6).** The number of these matrices will be equal to the number of hidden layers minus one.

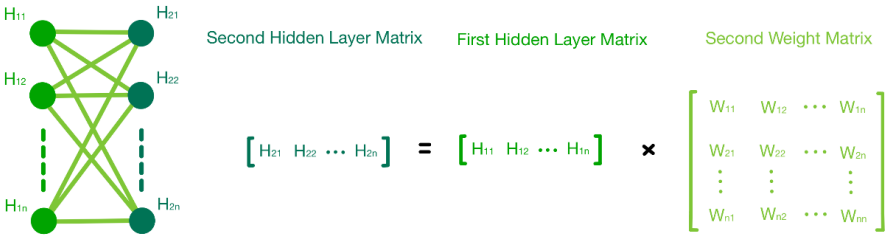

Fig. 9.6 Hidden layer to hidden layer matrix multiplication

Finally, the last matrix connects the last hidden layer with the outputs. The matrix has the same number of rows as the number of hidden neurons and two columns (number of outputs) **(see Fig. 9.7).**

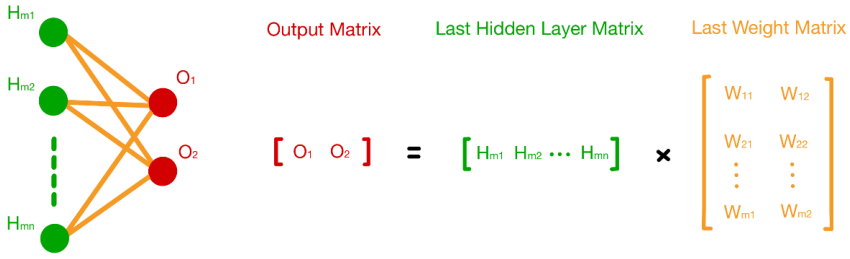

Fig. 9.7 Hidden layer to output layer matrix multiplication

The list of matrices is the same for all the simulation and the thing that differs from one neural network to another are the weight values. Initially these values are completely random and they will be modified by the genetic algorithm to improve the neural network.

The created neural network is used to obtain the two output values from the three inputs by using matrix multiplications and applying biases and non-linear functions as explained in the theory section. The following section extracted from the code is responsible of calculating the outputs as explained:

```
// calculates the first hidden layer and applies a hyperbolic
tangent
        hiddenLayers[0] = ((inputLayer * weights[0]) +
biases[0]).PointwiseTanh();

// calculates the remaining hidden layers and applies a
hyperbolic tangent to each
        for (int i = 1; i < hiddenLayers.Count; i++){
            hiddenLayers[i] = ((hiddenLayers[i - 1] *
weights[i]) + biases[i]).PointwiseTanh();}
```

```
// calculates the outputs and applies a hyperbolic tangent
        outputLayer = ((hiddenLayers[hiddenLayers.Count-
1]*weights[weights.Count-1])+biases[biases.Count-
1]).PointwiseTanh();
```

9.1.3. Genetic algorithm function

Finally, the genetic algorithm is controlled by a .cs file named Genetic Manager. The Algorithm is controlled by the following public variables:

- initialPopulation: indicates the number of neural networks that will compose the first population randomly generated.

- mutationRate: a float that ranges from 0 to 1 indicating the probability of mutation.

- bestAgentSelection: indicates the number of networks with the highest fitness score that will be selected.

- worstAgentSelection: indicates the number of networks with the lowest fitness score that will be selected. These will be used for the Crossover but will not survive to the next generation.

- numberToCrossover: indicates the number of neural networks that will result from the crossover.

These variables can be easily modified and adjusted to make the algorithm faster for each situation. These adjustments will be tested later on to see their effect on the algorithm efficiency. Moreover, there are two public variables (currentGeneration and currentGenome) that show the current Generation and Genome when the Lap Time Simulator is running. Just for clarification, a Genome is the Neural Network and goes from 0 to initialPopulation-1.

When starting the program, the Genetic Manager creates an initial population of randomly generated neural networks and will assign to the car the first network. Then, the car will run the circuit using the current network until it collides. When this happens, the manager checks if the network is the last of the current generation. If the network is not the last the manager just sends to the car the next network, but if it is, the manager starts repopulating.

The repopulation function executes the basic steps of a genetic algorithm as explained in Genetic algorithm section. First sorts the population according to their fitness score calculated by the following formula:

```
overallFitness = (totalDistanceTravelled * distanceMultiplier) +
(avgSpeed * avgSpeedMultiplier) + (((aSensor + bSensor + cSensor)
/ 3) * sensorMultiplier);
```

Where:

DistanceMultiplier, avgSpeedMultiplier and sensorMultiplier are public variables to manually specify the weight of each performance (i.e. how important is the distance travelled before colliding, the average speed and the distance between the car and the walls)

Then, the best and worst networks are selected (the number of networks selected depends on bestAgentSelection and worstAgentSelection variables). After, the selected networks are reproduced between them. This crossover just randomly selects a weight/bias value from one of the fathers for each new weight/bias to create a new child network. Each crossover uses two parents and generates two children. Then, the best networks and the new children go through mutation which with a certain probability (mutationRate) changes a weight value to a new random value. Finally, the new population is filled with new random networks and the car is sent the first network.

9.1.4. Flow diagram

As stated above, the program is divided in three different files (Car Controller, Neural Network and Genetic Manager). The Car Controller contains all the car variables and is responsible of the car's movement by using external neural networks to calculate the power and direction inputs. The Neural Network file creates a random neural network with a specified number of layers and neurons. Finally, the Genetic Manager applies the genetic algorithm to the neural networks in order to obtain a better result.

The flow diagram **(see Fig. 9.8)** illustrates the correlation between the 3 files. First, the car controller sets all the initial variables and initializes the neural network sending two user defined variables: layers and neurons. Then, the neural network file creates a neural network of the specified size, filled with random weight values, and sends this created network to the car controller.

After the initial settings, the car controller enters the FixedUpdate() function, which runs 50 times each second. This function obtains the car' sensors values and sends them to the created neural network. The network uses the 3 inputs and returns a value from 0 to 1 for the selected power and direction. Then, this 2 variables are used by the car controller to move the car using the MoveCar() function (This function applies the formulas from Car physics section).

Finally, the car controller updates the fitness value of the current network and checks if the car has lost the control by going out of the road or by skidding. If the car can continue, the FixedUpdate() function repeats and if not, the car resets and the network is saved with its results for the genetic manager.

Without the genetic manager the neural networks would be generated randomly and the probability to obtain a good driver would be extremely low. Therefore, the

genetic manager applies a genetic algorithm to improve the neural network and obtain an efficient driving car.

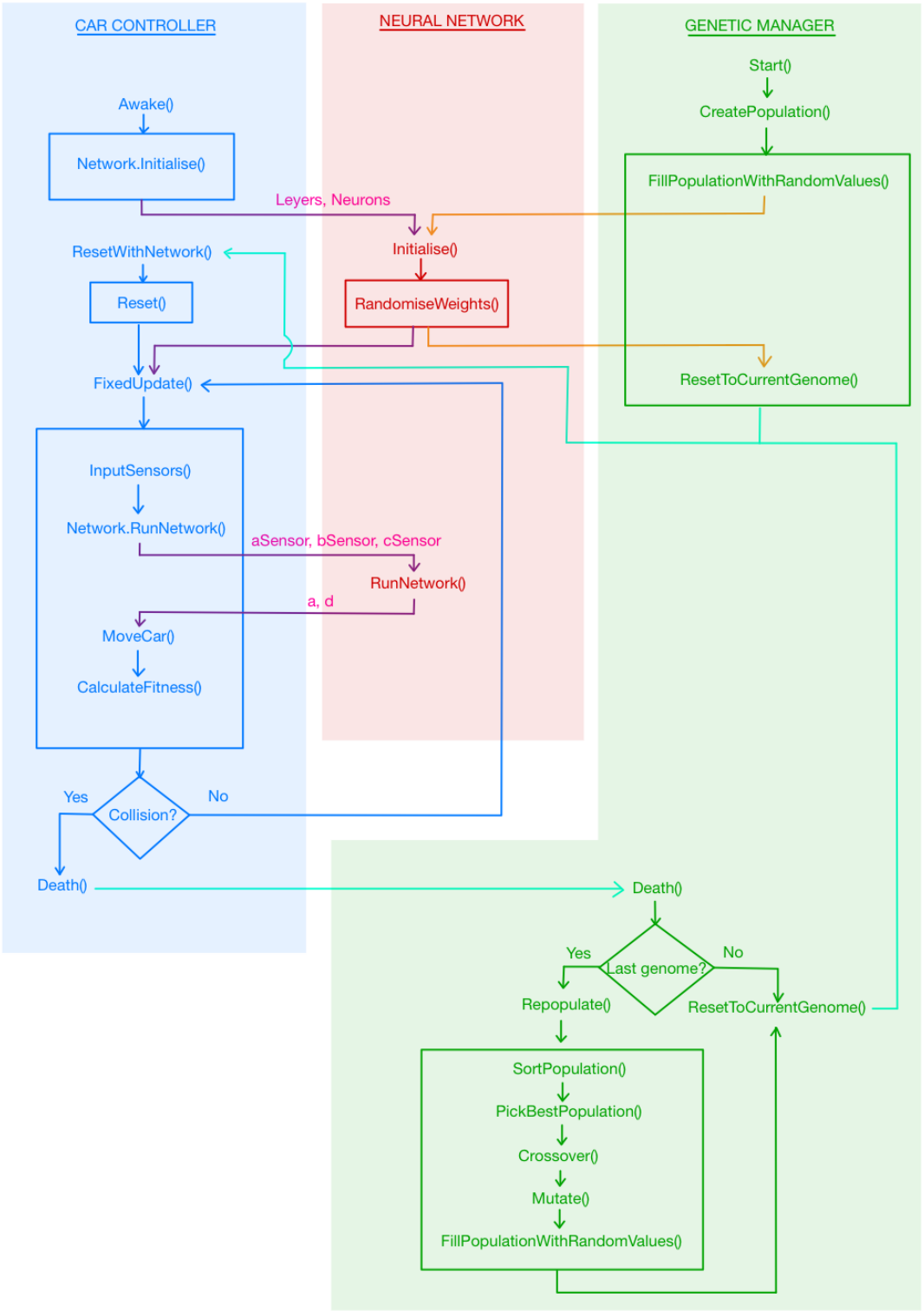

Fig. 9.8 Lap time simulator flow diagram (only essentials)

9.2. Testing

After developing the program, it is important to ensure that every function is working as it should. This section summarizes some of the tests performed to check the different features.

It is important to note that, due to the nature of this program, most of the test were done visually by checking if the car is doing what it should and by watching the numbers during the live simulation. Therefore, there are not a lot of test that can be done to automatically check if something is working fine or not.

The first created for testing is a simple track in which the car will run before testing the real competition circuits. The track is created on blender and consist of turns to the left and to the right **(see Fig. 9.9).** This way the car will be able to learn how to turn both sides.

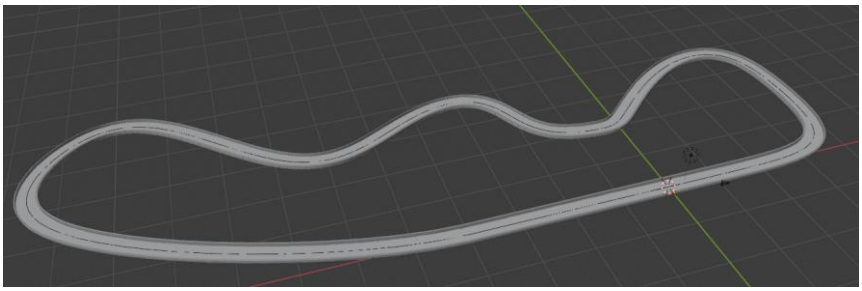

Fig. 9.9 3D model of a simple track created in blender

After importing the car into unity and downloading a low-poly car, it is time to test the death function (restarts the car on collision), the input sensors (3 rays that measure the distance from the cars to the walls. This will be the inputs to the neural network **–see Fig. 9.10-**) and the fitness function (just if the score if being calculated correctly).

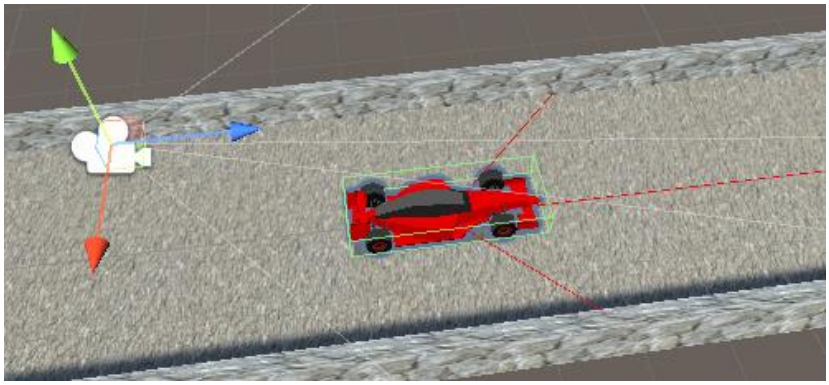

Fig. 9.10 unity scene with collision box around the car (green box) and input sensors (red rays)

At this point there is still no neural network, genetic algorithm or proper car physics. Instead, the acceleration and direction are introduced manually into the following provisional move function (the values in the function were obtained experimentally as it is only a test function):

```
  private Vector3 movement;
    public void MoveCar (float a, float d) {
        movement = Vector3.Lerp(Vector3.zero,new
Vector3(0,0,a*12.0f),0.02f);
        movement = transform.TransformDirection(movement);
        transform.position += movement;
        transform.eulerAngles += new Vector3(0, (d*90)*0.02f,0);
}
```

Where a is the input acceleration and d is the input direction.

Once everything is working correctly, it is time to test the neural network and apply it to the car controller instead of driving manually. The neural network file consists mostly in matrix multiplication. Therefore, great part of the testing is debugging the code to ensure that the matrix size is correct and that the multiplication is being done as planned.

Then, the network is tested to ensure that it is calculating an acceleration and a direction value for the car controller. The section of the code responsible of this is the following:

```
    public (float, float) RunNetwork (float input1, float input2,
float input3)
    {
        inputLayer[0, 0] = input1;
        inputLayer[0, 1] = input2;
        inputLayer[0, 2] = input3;

        inputLayer = inputLayer.PointwiseTanh();

        hiddenLayers[0] = ((inputLayer * weights[0]) +
biases[0]).PointwiseTanh();

        for (int i = 1; i < hiddenLayers.Count; i++)
        {
            hiddenLayers[i] = ((hiddenLayers[i - 1] * weights[i]) +
biases[i]).PointwiseTanh();
        }

        outputLayer = ((hiddenLayers[hiddenLayers.Count - 1] *
weights[weights.Count - 1]) + biases[biases.Count -
1]).PointwiseTanh();

        //acceleration [0-1] and direction [0-1]
```

```
        return (Sigmoid(outputLayer[0,0]),
(float)Math.Tanh(outputLayer[0,1]));
    }

    private float Sigmoid (float t)
    {
        return (1 / (1 + Mathf.Exp(-t)));
    }
```

This function just multiplies the matrixes as explained in Neural network function section. Note that in this first version for testing, acceleration is always positive but when applying the car physics it will have the possibility to be negative in order to brake.

Finally, the genetic algorithm needs to be tested. First, it is important to ensure that simple functions such as the mutation or the sorting loop are working. This functions can be tested with smaller networks.

After ensuring every function is working, the whole algorithm needs some test to check if it is doing what it should. The best way to test if it works correctly is by printing the fitness score and if it improves over the generations. Running the simulator with a test configuration **(see Fig. 9.11)** until it reaches a fitness score of 1000 resulted in **Fig. 9.12** (numerical results in Table C.1).

Fig. 9.11 Genetic algorithm public variables configuration for testing

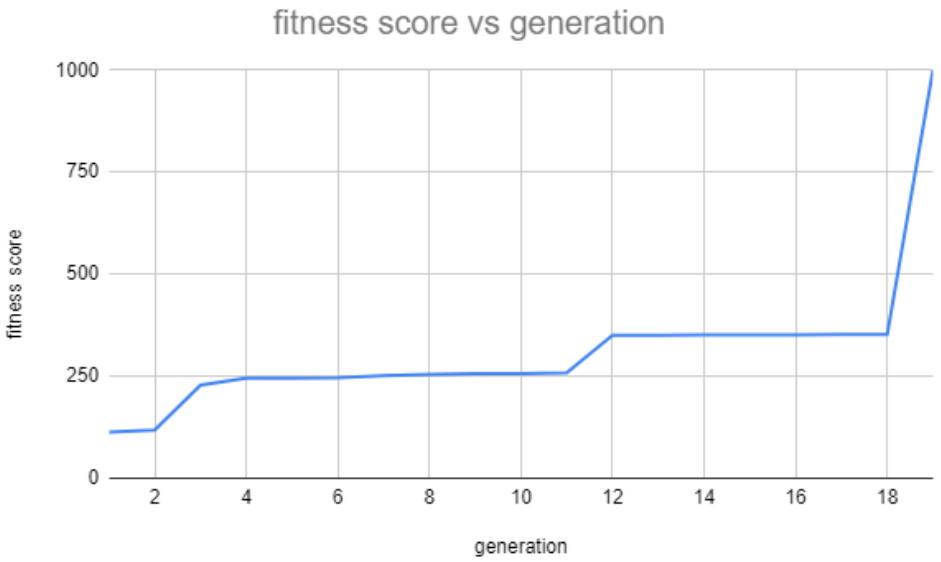

Fig. 9.12 Fitness score vs number of generations graph

The lap time simulator learnt to drive until the end of the first right corner in just 3 generations. After 8 generations it reached midway through the second corner (first left corner). In 12 generations it reached the fourth corner and in only 19 generations the simulator reached the maximum score of 1000 which is equivalent to two full laps to the track.

The program is working as expected and in the following chapter it will be improved to teach it to drive faster, add extra implementations and optimize the program. But first, the provisional car movement needs to be replaced by the actual car physics. To do so, the car physics need to be tested too.

To test all the car physics, a test mode is implemented to the program. If the test mode checkbox is checked, on play the car will be transported to a big plane and the acceleration and director inputs will be controlled by the user. Therefore, the neural network and genetic algorithm files do not interfere. Moreover, in the unity interface all the important car parameters are shown live to let the user see if everything is fine **(see Fig. 9.13).**

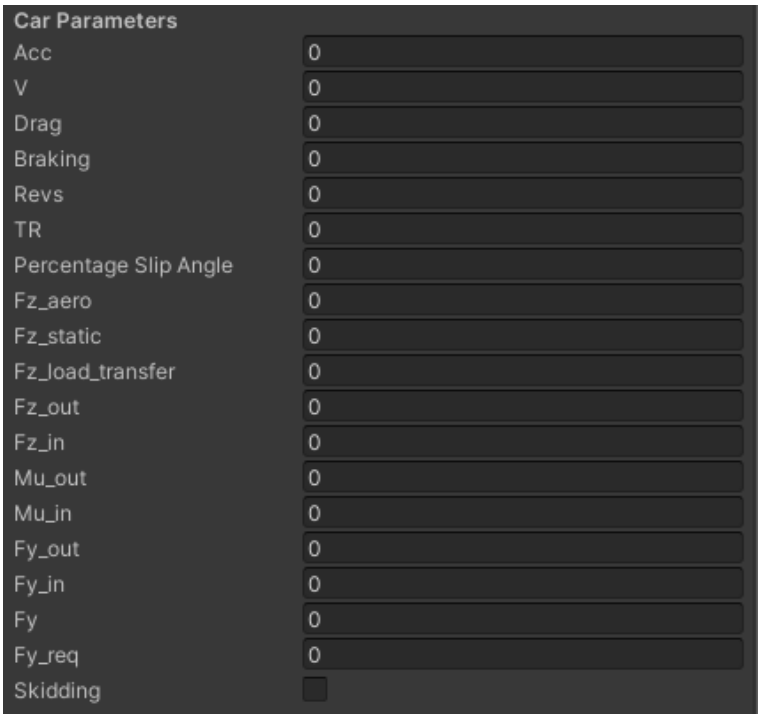

Fig. 9.13 Car parameters in unity interface

The first thing to test is the acceleration and braking. To do so, random values of acceleration are selected and, using the formulas manually, calculate if the values displayed by the program are correct.

For turning the methodology is the same. But moreover, here is important to see how the car is moving in the scene. By doing so, some errors were found and corrected in the original formulas. Furthermore, the need of a skid function was also found because the car was able to turn at unrealistically high speeds.

Testing the car physics is the most time consuming test because it needs to be done manually. But after completing the tests it is ready to be added to the full program and jump into the next chapter.

10. EXPERIMENTATION

Once the simulator is able to find a neural network capable of driving without colliding, it is time to improve the simulator performance and add some extra implementations.

This chapter describes the different performance improvements implemented to the simulator after testing it. And a section dedicated to extra implementations that help visualize the data and manage the simulations.

10.1. Improvements

After implementing the car physics to the working first version of the lap time simulator, it seemed to have more difficulties improving the network. This is because the movement now is not linear as for the test version. This section covers the different changes that are made to the program to work with the car physics more efficiently.

10.1.1. Genetic algorithm optimization

First of all, the simulator usually gets stuck at the first turn. It learns to turn to the first corner direction but then it has difficulties learning to turn to the other direction.

The problem causing this is the network complexity. The neural network is too simple for the program to learn to drive with the new car physics. The solution implemented is to augment the hidden layers from 1 to 2 adding 10 new hidden neurons.

Moreover, the crossover is made between the best and worst agents of each population. After some experimentation, the worst agents were eliminated from the crossover. The original idea was to reproduce the best with the worst to add more randomness into the new generation, but this is already implemented by the mutation and therefore it is not necessary. The changes can be seen in the following figure **(see Fig. 10.1):**

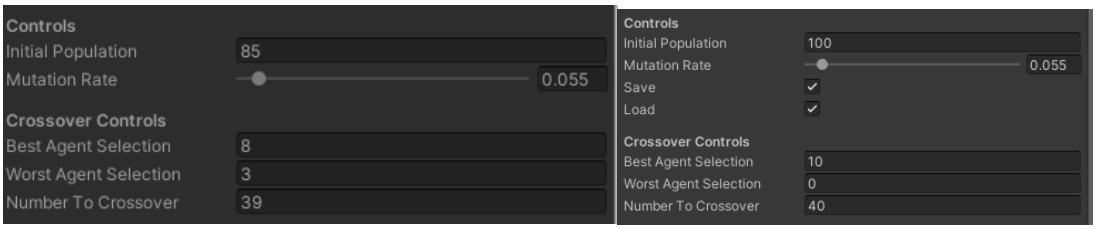

Fig. 10.1 Test genetic algorithm configuration (Left) and new configuration (Right)

10.1.2. Variable mutation rate

In addition to the previous changes in the configuration, the mutation rate needs to be incremented to explore new possibilities in the new generations. A new function is implemented to increment the mutation rate if the best agent of the current population is not better than the previous best (i.e. the network has not improved from the previous generation).

Note that the mutation rate also has a maximum value because a large mutation rate would erase the network and create a new random one.

10.1.3. More inputs

In concordance to the previous changes, the number of inputs is also augmented to increase the network complexity.

First, the number of sensors is increased by adding two new lateral sensors to the 3 existent ones **(see Fig. 10.2)** to increase the environment awareness. Moreover, the sensor function is improved to return a unitary value (i.e. all the sensors values are now relative to the maximum value which will be 1). During long straights, the central sensor returned a very large value and caused the values at corners to be very small and not significant to networks that learned to drive straights.

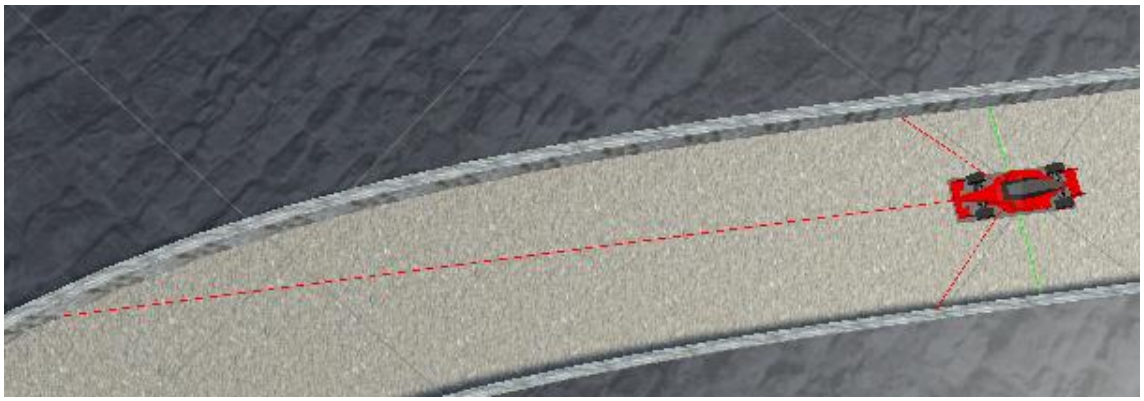

Fig. 10.2 Original 3 sensors (red) and the 2 new lateral sensors (green)

In addition, the velocity parameter is added to the inputs making a total of 6 inputs. The velocity is a crucial input because it is not the same reaction to turn a specific corner at low speed than at high speed.

10.1.4. Speed mode

With the previous improvements the simulator is able to find a network that is able to run forever without colliding. Now the objective is to find the network that can complete the circuit faster. To do so, a new mode is created.

This new mode called “speed mode” gets activated automatically when a neural network reaches a fitness score of 1000, which is equivalent to completing two approximately two laps.

When speed mode is activated, the average speed multiplier gets significantly increased and the crossover parameters are set to reduce the number of randomly generated networks. The new population will be filled by reproducing the current best networks (i.e. the next generation will most likely now how to drive too).

This implementation resulted in some erroneous strategies (e.g. the car tries to cheat and increase the average speed by doing wider turns and by wriggling on the main straight to increase the travelled distance and maintaining high speeds).

The solution is to add a finish line to count the laps the car is doing and calculate the real lap time. The fitness formula is now calculated when the car completes a determined number of laps (e.g. 5 laps) with the average lap time of the car. With this mode the simulator is able to find the best lap time for a determined circuit **(see Fig. 10.3)** (numerical results in Table C.2 and Table C.3).

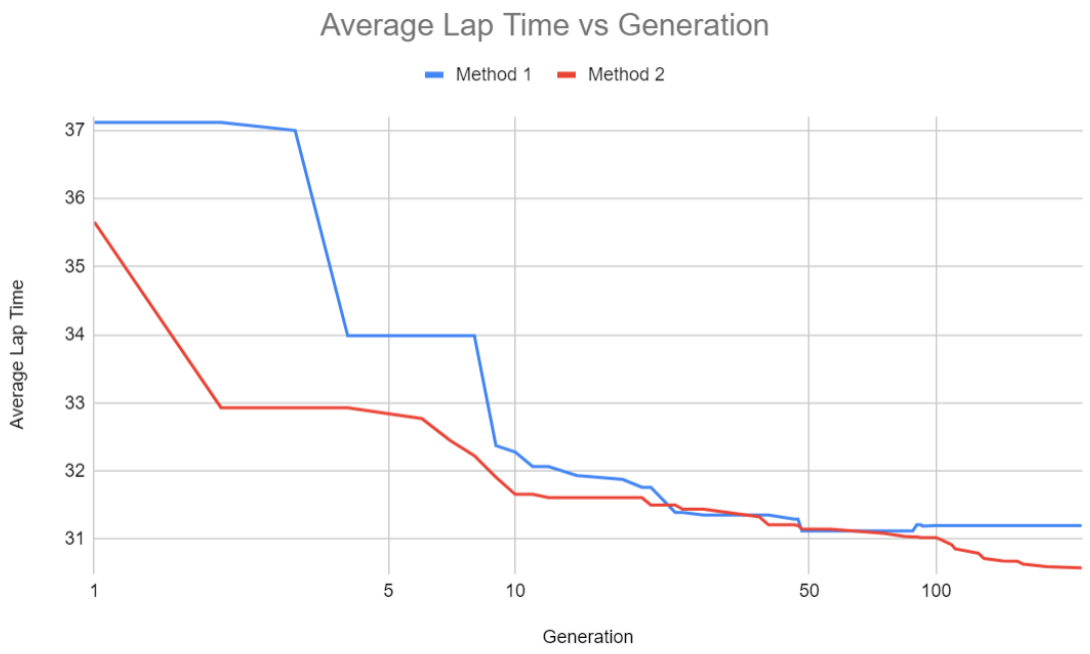

Fig. 10.3 Speed mode test comparison between method 1 (increase speed multiplier) and method 2 (use average lap time to calculate fitness)

10.2. Extra functions

This section explains the main implemented functionalities that do not change the way the simulator works but are essential for a better interaction between the user and the simulator.

10.2.1. HUD (Heads-Up Display)

Up to this moment, all the variables can only be seen by using the console or by selecting a specific component to see its public variables on the unity interface. In order to help visualize all the necessary information in a single screen, a HUD is created with live information and a slider to control the time scale without exiting the simulator screen **(see Fig. 10.4).**

Fig. 10.4 HUD example

10.2.2. Save and load functions

Simulations can take quite a few generations to complete, and even more to reach the perfect lap time, this is why a way to save the best network for a later use is needed.

The save function stores the network properties (number of inputs, outputs, hidden layers and hidden neurons) and the weight and bias values of the current best network. This way, when a new network beats the current best fitness score, the current networks is automatically saved into a txt file.

The load function reads a saved txt file and translates the information to create the network. This loaded network is used as the first genome of the first generation and will propagate naturally to the next generations.

Note that during reproduction, if a network has a significantly higher fitness, it will be reproduced with a higher chance than the new random networks generated in the first generation. But if the loaded network does not have a high fitness it will reproduce similar to the other new networks.

This save and load functions can be used to load a network that knows how to drive without colliding and automatically trigger the speed mode and start optimizing the velocity just from the start of the simulation.

11. FINAL RESULTS

Finally, it is time to apply the aerodynamic results obtained in FSAE AERODYNAMICS DESIGN to the lap time simulator and test its efficiency in a racing track.

The simulations in this chapter start from the network obtained in the previous chapter using the Save and load functions to reduce the simulations duration and activate the Speed mode from the start. The network loaded was obtained after 189 generations using test aerodynamic parameters. The lap time of this network is not important here as the comparison with these results has no sense, but the network has learnt to drive the circuit fast and this will be useful even though the aerodynamic parameters of the car change.

The car specifications used in the simulations can be seen in **Fig. 11.1**. And the variables will be the drag value (Dref) and the lift value (Lref). Note that most of the specifications are reference values that should be modified according to the car specifications after building it.

```
//Car Data
private float m=750f; //Vehicle mass [Kg]
private float P=10f; //Engine Power [Kw]
private float Cr=0.02f; //Rolling Coeficient
private float muTyre=1.4f; //Friction Coeficient when braking
private float maxSlipAngle=6f; //maximum slip angle[º]
private float hcg=0.27f; //Gravity center height [m]
private float track=1.54f; //Car width [m]
private float maxTurnAngle=20.0f; //Max turning angle [º]
private float wheelbase=2.0f; //distance between front tyres and rear tyres [m]

//Aero Data from the simulations
private float vref=13.89f; //Reference Velocity [m/s]
private float Dref=0.118f; //Reference Drag [KN]
private float Lref=0.159f; //Reference Downforce [KN]
```

Fig. 11.1 Car specifications for the lap time simulator

11.1. Car without aerodynamic devices

A car without any aerodynamic study should have an almost null Lift force. This is because the forces created by each of the car component should cancel each other. Therefore, the lift used for these simulations is L = 1 N, which is an almost negligible force. And for the drag a good approximation is a drag of 200 N at 50 km/h, which is a bit more that the obtained from the designed car. These values can be justified because, the missing bodywork and undertray, would cause the air to enter the chassis, between the motor's admission and scape tubes, creating a large amount of drag. Note that these aerodynamic values are just for reference and should be replaced with real simulation results for an accurate measurement.

After running the simulator for 6 hours, the average lap time over 5 laps improved from an initial 31.8 seconds to 31.5 seconds after 99 generations where the simulator stopped improving the time **(see Fig. 11.2)** (the numerical results can be found in Table C.4).

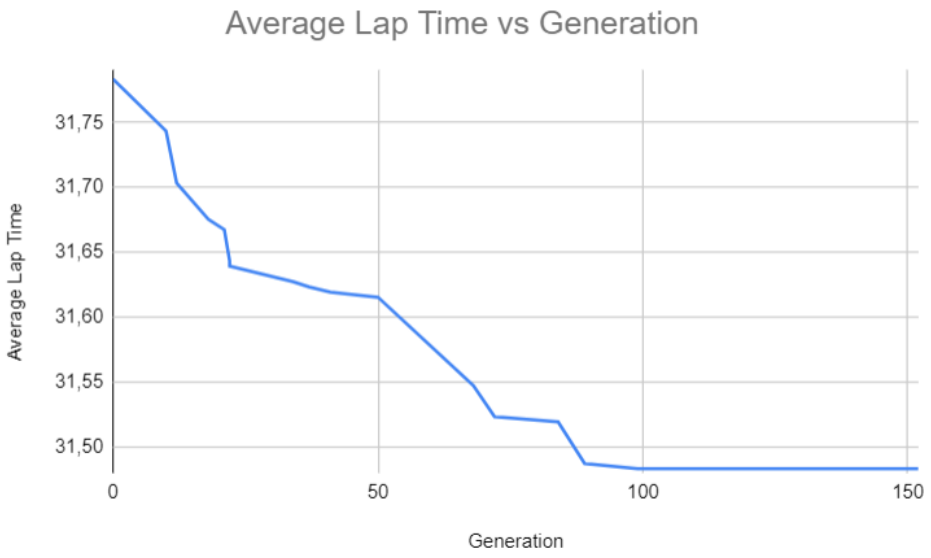

Fig. 11.2 Average lap time simulation with a non-aerodynamic car plot

11.2. Car with aerodynamic devices

For this simulation the aerodynamic parameters are the obtained in Final result section from FSAE AERODYNAMICS DESIGN chapter (L = 159 N and D = 118 N). This simulation starts with the same loaded network as the one used in the previous section.

After running the simulator for 8 hours, the average lap time over 5 laps improved from an initial 30.9 seconds to 30.6 seconds after 218 generations where the simulator stopped improving the time **(see Fig. 11.3)** (the numerical results can be found in Table C.5).

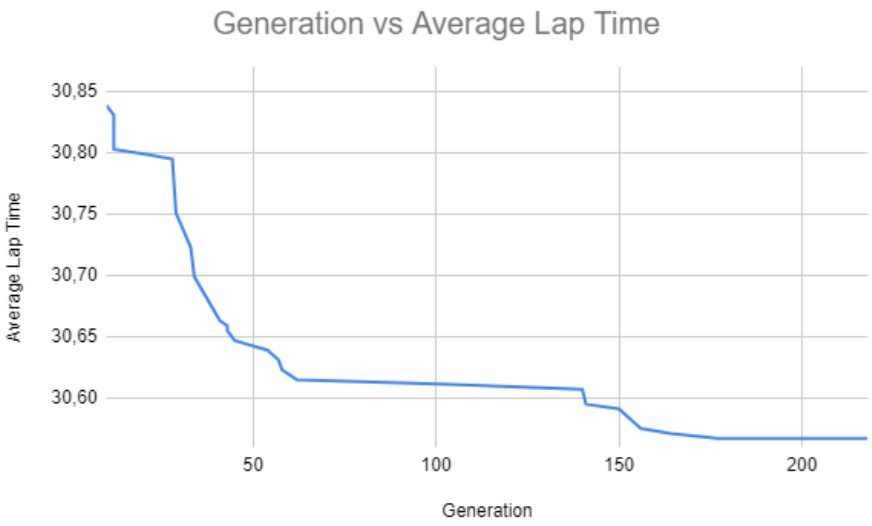

Fig. 11.3 Average lap time simulation with aerodynamic car plot

11.3. Comparison

Both simulations improve the average lap time by 0,3 s until reaching their best result **(see Fig. 11.4).** The difference between the non-aerodynamic car and the designed aerodynamic car is 0,916 s each lap which is equivalent to a 3% increase in performance for the designed one. A difference of one second in a 30 s circuit is significant in racing but a better comparison should be done using real car parameters to obtain more accurate results. In this particular case the car has been able to drive through the track with a low use of lift during corners.

.

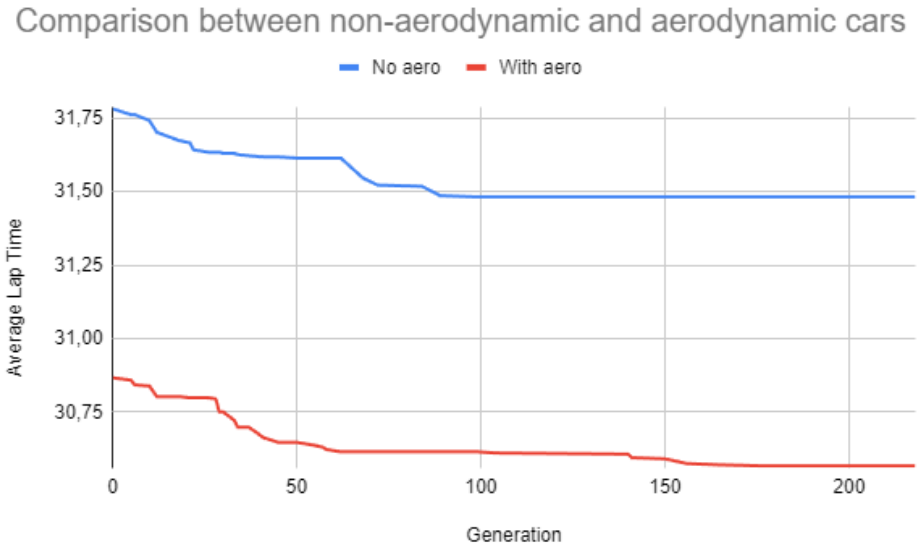

Fig. 11.4 Comparison between non-aerodynamic and aerodynamic cars

12. CONCLUSION

12.1. Project retrospective

Overall, most of the objectives proposed regarding the car design has been achieved. The design follows all the competition rules and the CFD simulations verified that the aerodynamic impact of the designed devices is positive and therefore the past season car has been improved. The car generates a lift force of -159 N and a drag force of 118 N despite the undertray not generating the lift that it should, and the angles of attack for all the airfoils being able to be further optimized.

Regarding the building and compatibility, all the devices have been built except the front and rear wings. But the reason of not building them has been team tight schedules and the designs will be saved for the next season.

To a large extent, the work plan has been followed but the time scheduled for some task have been prolonged. The mesh refinement took much longer than expected and this delayed the simulations.

On the other hand, the lap time simulator goals have been successfully completed. The realistic physics implementation has been the most difficult task because of the complexity of summarizing them into a few formulas able to run fast enough to not slow the simulator.

The genetic algorithms and neural network reduce the simulation time by an infinite amount. Just take for example the network size used for the last chapter. It has 6 inputs, 10 hidden neurons, 2 hidden layers and 2 outputs. This is equivalent to 202 variables and the simulator has been able to optimize them in 6 to 8 hours. If not using a genetic algorithm, the number of possible combinations that should be tested is 10^{2020} (limiting each variable to 10 decimals). This amount of test is impossible to do nowadays. Therefore, the objective has been successful.

12.2. Personal assessment

While doing this project I learned the complexity of aerodynamics in racing cars and helped me expand my knowledge obtained from the university about aerodynamics. Also, the experience of designing the aerodynamic devices for the team, with the constant reunions with my department and other departments, helped me realize the importance of organization in big engineering teams.

On the other hand, the process of making the lap time simulator made me discover the infinite possibilities of deep learning and made me anxious to start another related project.

If I had the opportunity to give some guidelines to someone starting this project, I would suggest to focus the aerodynamic part in designing just one or two devices (e.g. the front and rear wings) instead of the whole car. This will allow to focus more simulation time to each device. Moreover, I would recommend, if possible, to try another CFD software such as Ansys, to make the mesh refinement easier and faster.

12.3. Future work

Implementing some of the ideas explained in chapter 8 would be a good way to continue the development of the project and improve the designs. Probably, the best way to start would be to implement the Gurney flap in the wings, test a DRS system or add some Vortex generators over the front wing and bodywork, since these are the faster and easier options to develop. Aerodynamics can also be improved by conducting more simulations and optimizing the angle of attack of the wings. As previously stated, aerodynamics is constantly evolving and there is always room to improvement. The only requisite is time.

For the lap time simulator some improvements can also be implemented to speed up the process. First of all, modifying the program to run multiple agents simultaneously would probably speed up the code. This implementation would make it easy to compare visually the different networks and discard them faster, but it will still be limited to the computational power (similar to increasing the time scale in the current version). Another idea is to try some reinforced learning techniques such as for example, dividing the circuit in multiple sections and compare the times each network needs to complete each section (e.g. it can be the case that the best current network is doing okay in sectors 2 and 3 but there are other networks that are best at the first sector).

The last thing that would help improve the project is to build the car and do real circuit and aerodynamic test. This would be the best way to validate the aerodynamic results and adjust the lap time simulator to be as accurate as possible.

In a nutshell, even though most of the project objectives have been fulfilled, it does not mean the project is over. This could be used as a starting point for more complex developments and hopefully someone will make use of it one day.

BIBLIOGRAPHY

[1] PAKKAM, SRIRAM SARANATHY. *High Downforce Aerodynamics for Motorsports*, [Online Resource] Available: https://repository.lib.ncsu.edu/bitstream/handle/1840.16/7256/etd.pdf?sequence=2&isAllowed=y [Last Access: 26/08/2022]

[2] Solidworks Official Website, [Website] Available: https://www.solidworks.com [Last Access: 9/09/2022]

[3] Solidworks Flow Simulation Official Website, [Website] Available: https://www.solidworks.com/product/solidworks-flow-simulation [Last Access: 9/09/2022]

[4] Unity Official Website, [Website] Available: https://unity.com/ [Last Access: 9/09/2022]

[5] Blender Official Website, [Website] Available: https://www.blender.org/ [Last Access: 9/09/2022]

[6] Solidworks Logo, [Image] Available: https://worldvectorlogo.com/es/logo/solidworks-logo-1 [Last Access: 9/09/2022]

[7] Unity Logo, [Image] Available: https://worldvectorlogo.com/es/logo/unity-technologies-logo [Last Access: 9/09/2022]

[8] Blender Logo, [Image] Available: https://worldvectorlogo.com/es/logo/logo-blender [Last Access: 9/09/2022]

[9] Formula Student competitions in Europe, [Image] Available: https://motorsportvng.upc.edu/es/vilanova-formula-team/competicion [Last Access: 20/06/2022]

[10] Vandellós Ripoll, J., "*Estudio y diseño aerodinámico de un monoplaza de formula student"*, UPC, 2020.

[11] VFT racing car Nova 6, [Image] Available: https://motorsportvng.upc.edu/ca [Last Access: 9/09/2022]

[12] Formula Student, *Formula Student Rules 2022*, [Online Resource] Available: https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FS-Rules_2022_v1.0.pdf [Last Access: 26/08/2022]

[13] John D. Anderson, Jr, *Fundamentals of Aerodynamics Sixth Edition*, Mc Graw Hill, [Online Resource] Available: https://aviationdose.com/wp-content/uploads/2020/01/Fundamentals-of-aerodynamics-6-Edition.pdf [Last Access: 26/08/2022]

[14] Kevin Gurney, *An Introduction to Neural Networks*, University of Sheffield, [Online Resource] Available: https://www.inf.ed.ac.uk/teaching/courses/nlu/assets/reading/Gurney_et_al.pdf [Last Access: 26/08/2022]

[15] The logistic curve, [Image] Available: https://medium.com/@shiny_jay/logistic-regression-a9a8749e1e68 [Last Access: 9/09/2022]

[16] Valero, Miguel, "*Introduction to Genetic Algorithms for Combinational Problems"*, UPC, 2021.

[17] Gantt Diagram utility, [Website] Available: https://www.onlinegantt.com/#/gantt [Last Access: 28/08/2022]

[18] Hirsch, C. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier, 2007 [Online Resource] Available: https://cfd.spbstu.ru/agarbaruk/doc/Hirsch%20C.%20Numerical%20Computation%20of%20Internal%20and%20External%20Flows.Volume1-Fundamentals%20of%20Computational%20Fluid%20Dynamics(Elsevier,2nd%20edn,2007).pdf [Last Access: 26/08/2022]

[19] Solidworks, *An Introduction to Neural Networks*, Haw Ridge Systems, [Online Resource] Available: https://pages.hawkridgesys.com/rs/499-VQL-975/images/enhanced-turbulence-modeling-solidworks-flow-simulation-whitepaper.pdf [Last Access: 26/08/2022]

[20] Pope, Stephen. *Turbulent Flows*, 2000 [Online Resource] Available: https://www.usb.ac.ir/FileStaff/8477_2018-10-15-13-52-48.pdf [Last Access: 26/08/2022]

[21] SolidWorks Flow Simulation Online User's Guide, Solidworks, 2021.

[22] Glenn Research Center, Navier Stokes Equations: 2 dimensional unsteady, NASA [Online Resource] Available: https://www.grc.nasa.gov/www/k-12/airplane/nsegs.html#:~:text=This%20area%20of%20study%20is,dependent%20conservation%20of%20energy%20equation [Last Access: 26/08/2022]

[23] Valerio. Newtonian Fluid Example [Online Resource] Available: https://physics.stackexchange.com/questions/293349/intuitive-definition-for-newtonian-fluid [Last Access: 26/08/2022]

[24] Flow vectors for positive (Left), negative (Middle) and neutral Divergence (Right), [Image] Available: https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/divergence-and-curl-articles/a/divergence [Last Access: 9/09/2022]

[25] Car orientation, [Image] Available: https://www.kindpng.com/imgv/ToxxRTJ_formula-1-car-top-down-png-download-transparent/ [Last Access: 9/09/2022]

[26] FX74_CL5_140 (Left), S1223 (center), LNV109A (Right), [Image] Available: http://airfoiltools.com/plotter/index [Last Access: 9/09/2022]

[27] DRS closed and open comparison, [Image] Available: https://www.presticebdt.com/how-f1-drs-works-the-aerodynamics-of-formula-one-drs-explained/ [Last Access: 9/09/2022]

[28] Aztec Racing. *Project DRS: A Drag Reduction System developed for the Formula SAE*, San Diego State University [Online Resource] Available: https://zaguan.unizar.es/record/61248/files/TAZ-TFG-2016-2705.pdf [Last Access: 26/08/2022]

[29] David J. Penner. *Aerodynamic Design and Analysis of a Formula SAE Drag Reduction System (DRS)*, University of Manitoba [Online Resource] Available: https://www.researchgate.net/publication/339997626_Aerodynamic_Design_and_Analysis_of_a_Formula_SAE_Drag_Reduction_System_DRS [Last Access: 26/08/2022]

[30] Formula *one Front Wing Optimization and Configuration: Modelling the Bending and Lift of a F1 Front Wing*, American University of the Middle East (AUM) [Online Resource] Available: https://www.ieomsociety.org/ieom2018/papers/667.pdf [Last Access: 26/08/2022]

[31] Example of a flexible wing tilt, [Image] Available: https://espanol.motorsport.com/f1/news/alerones-flexibles-prohibidos-formula-1/4812953/ [Last Access: 9/09/2022]

[32] Giorgio Piola sketch of the JS11 hidden flaps, [Image] Available: https://www.motors-addict.com/en/article/formula1/revealed-how-to-spot-what-an-f1-team-is-trying-to-hide/5dc9dea4493e8b5f7d5ffd00?utm_campaign=post-auto&utm_medium=facebook&utm_source=f1-addict-au [Last Access: 9/09/2022]

[33] Gurney flap, [Image] Available: https://www.graining.es/tecnica-f1-gurney-flaps/ [Last Access: 9/09/2022]

[34] Sam Prabhu Jesu Rajendran. *How Dimples on a Golf Ball Affect Its Flight and Aerodynamics*, Simscale Blog [Online Resource] Available: https://www.simscale.com/blog/how-dimples-affect-golf-ball/ [Last Access: 26/08/2022]

[35] Golf ball aerodynamics (right) compared to a smooth ball (left), [Image] Available: https://www.researchgate.net/publication/312219667_International_Journal_of_Advanced_Production_and_Industrial_Engineering_INNOVATIVE_DESIGN_C

OMPUTATIONAL_MODELLING_Analysis_on_Drag_Reduction_of_Bluff_Body_using_Dimples [Last Access: 9/09/2022]

[36]. *Aerodynamically shaped vortex generators*, University of Denmark [Online Resource] Available: https://www.simscale.com/blog/how-dimples-affect-golf-ball/ [Last Access: 26/08/2022]

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APPENDIX

TÍTOL DEL TFG: FSAE aerodynamics and lap time simulator

TITULACIÓ: Doble titulació en Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació i Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Rubén Batanero Ramírez

DIRECTORS: Adeline de Villardi de Montlaur i Miguel Valero Garcia

DATA: 5 de setembre del 2022

APPENDIX A. CFD MESH REFINEMENT AND RESULTS

This appendix contains the CFD simulation results from the general mesh refinement and the later adjustments for each aerodynamic device.

Table A.1 Initial mesh refinement with multiple global and local mesh configurations

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
MSHD	8593560	875	44:57:23	2,81	-9,282	2	1,3	6	0,00044		
MSHD(1)	2144029	843	3:09:50	2,384	-5,779	2	1,3	4	0,01		
MSHD(2)	14569033	637	38:54:01	2,014	-7,541	2	1,3	5	0,01		
MSHD(3)	2458378	462	1:59:56	1,831	-5,681	2	1,3	4	0,01	0,023	0,0529
MSHD(4)	4930141	692	5:04:43	1,784	-7,915	2	1,3	4	0,02	0,046	0,1058
MSHD(5)	10803184	888	16:19:11	1,935	-8,91	2	1,3	4	0,04	0,092	0,2116
MSHD(6)	1554694	382	0:52:08	1,77	-6,548	2	1,3	3	0,04	0,092	0,2116
MSHD(7)	4412766	514	3:23:40	1,689	-6,323	2	1,3	3	0,1	0,23	0,529
MSHD(8)	10677626	524	10:09:07	1,691	-6,307	2	1,3	3	0,2	0,46	1,058
MSHD(9)	777533	276	0:15:11	1,915	-4,292	2	1	3	0,1	0,23	0,529
MSHD(10)	2018027	348	0:51:10	1,926	-4,515	2	1	3	0,2	0,46	1,058
MSHD(11)	1809216	374	0:49:28	1,792	-3,14	3	1	3	0,1	0,23	0,529
MSHD(12)	4765855	407	2:28:21	1,702	-3,607	3	1	3	0,2	0,46	1,058
MSHD(13)	3069124	451	1:43:55	1,632	-5,959	4	1	3	0,1	0,23	0,529
MSHD(14)	8108613	529	5:40:52	1,674	-5,174	4	1	3	0,2	0,46	1,058
MSHD(15)	6232782	552	4:37:26	1,753	-4,925	5	1	3	0,1	0,23	0,529

Table A.2 MSHD(8) mesh refinement (green: valid results; yellow and red: not valid results)

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
MSHD(8) A	9410969	733	12:32:12	1,785	-7,385	2	1,2	3	0,2	0,46	0,7
MSHD(8) B	4417546	433	3:57:36	1,634	-6,542	3	1,2	2	0,2	0,46	
MSHD(8) C	9.713.955	525	9:39:11	1,726	-6,329	4	1,2	2	0,2	0,46	
MSHD(8) D	6758088	498	6:39:42	1,644	-6,411	5	1,2	1	0,46		
MSHD(8) E	13344826	623	36:03:13	1,818	-7,796	6	1,2	1	0,46		
MSHD(8) F	12461403	931	112:50:57	1,943	-9,146	7	1,2	1	0,1		
MSHD(8) G	6248742	531	26:23:57	1,579	-6,397	7	1,2				

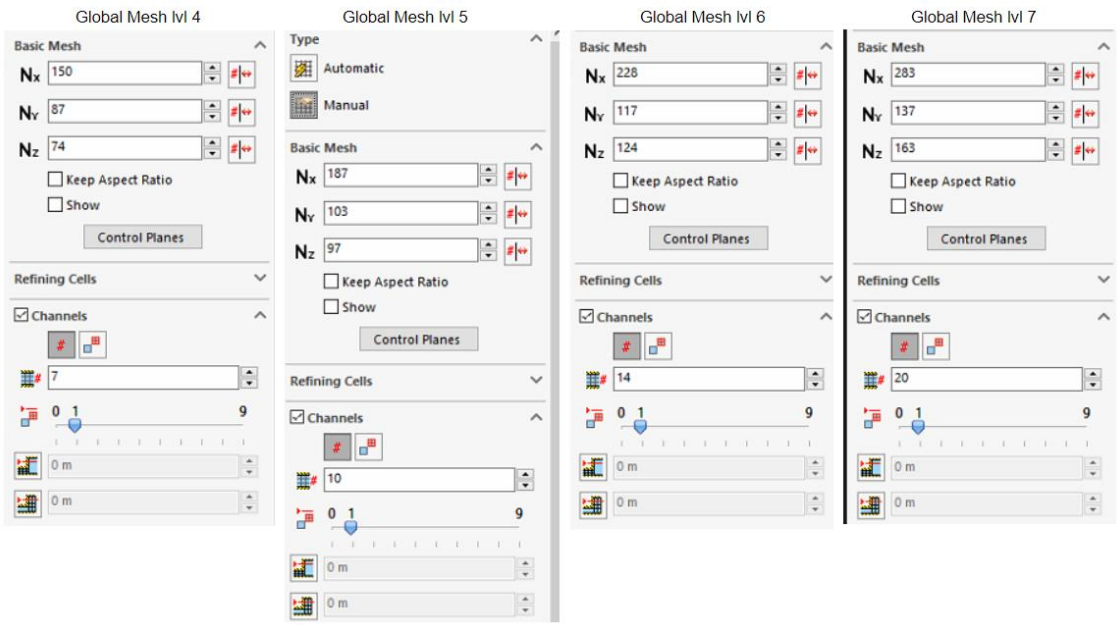

Fig. A.1 Settings for the global mesh levels used

Table A.3 Nose cone mesh refinement configuration and results

Nose cone

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	5549167	357	2:41:15	5,105	1,267	5	1,2	1	0,5		
Mesh 2	4886582	343	1:41:40	5,011	1,316	5	1,2	1	0,4		
Mesh 3	6317487	373	14:16:28	5,024	1,296	5	1,2	1	0,6		

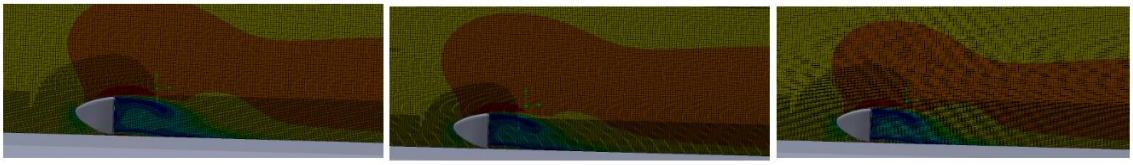

Fig. A.2 Nose cone mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

Table A.4 Undertray mesh refinement configuration and results

Undertay v3

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	6565018	444	5:00:34	17,294	-1,584	5	1,2	1	0,5		
Mesh 2	5756714	428	4:13:18	17,16	-1,793	5	1,2	1	0,4		
Mesh 3	7349802	469	6:18:54	15,562	-2,401	5	1,2	1	0,6		

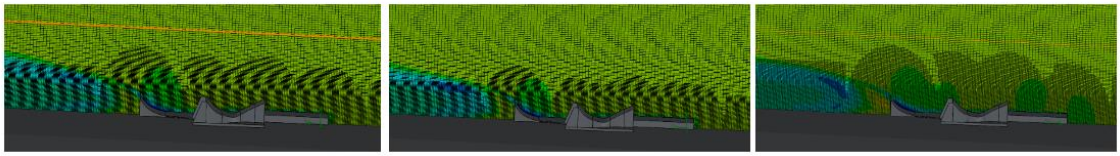

Fig. A.3 Undertray mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

Table A.5 Front wing mesh refinement configuration and results

Frontwing v3

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	5845006	362	3:14:09	16,337	-25,961	5	1,2	1	0,5		
Mesh 2	4971910	346	2:19:32	16,157	-26,982	5	1,2	1	0,4		
Mesh 3	6596120	377	3:48:35	16,168	-27,144	5	1,2	1	0,6		

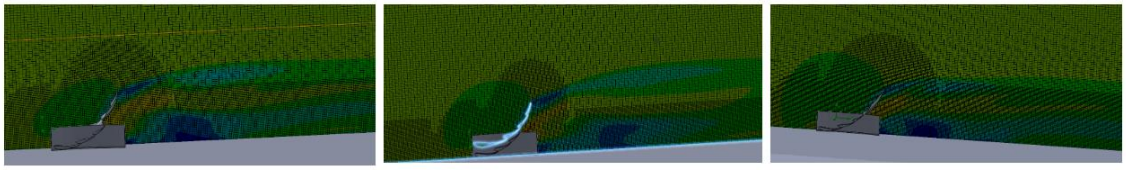

Fig. A.4 Front wing mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

Table A.6 Rear wing v1 mesh refinement configuration and results

MSHD with enplates v1

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	3870193	315	2:02:35	22,653	-64,462	5	1,2	1	0,46		
Mesh 2	4787746	338	2:49:21	22,833	-64,896	5	1,2	1	0,6		
Mesh 3	4928642	341	2:52:39	22,578	-64,971	5	1,2	1	0,62		

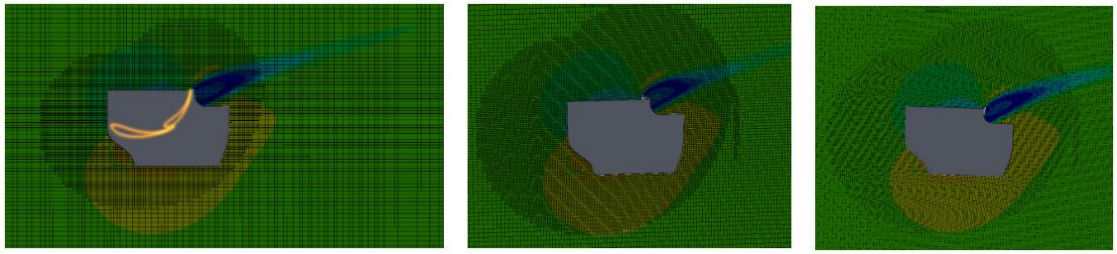

Fig. A.5 Rear wing v1 mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

Table A.7 Rear wing v2 mesh refinement configuration and results

MSHD with enplates v2

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	6228637	1473	23:42:31	31,503	-67,362	5	1,2	1	0,62		
Mesh 2	4053506	1277	7:54:47	29,364	-62,65	5	1,2	1	0,35		
Mesh 3	4344951	423	12:24:54	31,312	-70,281	5	1,2	1	0,39		
Mesh 4	5226629	349	4:22:09	30,443	-68,812	5	1,2	1	0,5		

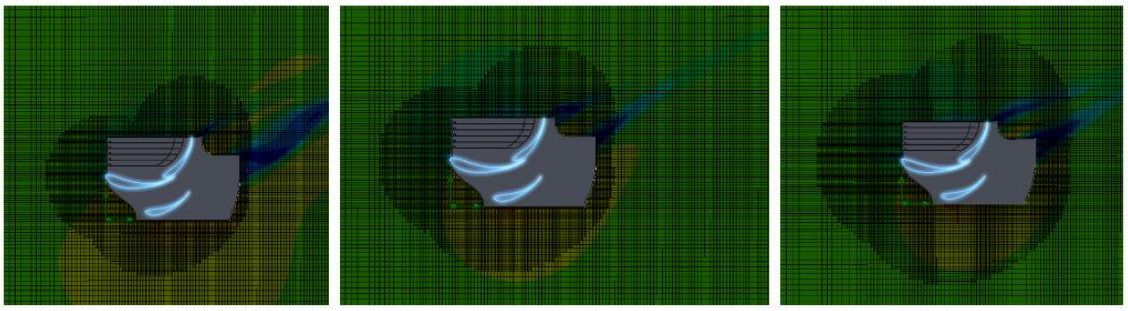

Fig. A.6 Rear wing v2 mesh refinement plots (mesh 2 –Left-, mesh 3 –center- and mesh 4 –right-)

Table A.8 Rear wing v3 Mesh Refinement configuration and results

MSHD with enplates v3-1

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	2726177	1118	4:07:42	28,379	-61,524	5	1,2	1	0,35		
Mesh 2	4011944	1011	5:42:26	28,671	-64,192	5	1,2	1	0,62		
Mesh 3	4315205	425	2:34:38	29,006	-67,881	5	1,2	1	0,67		

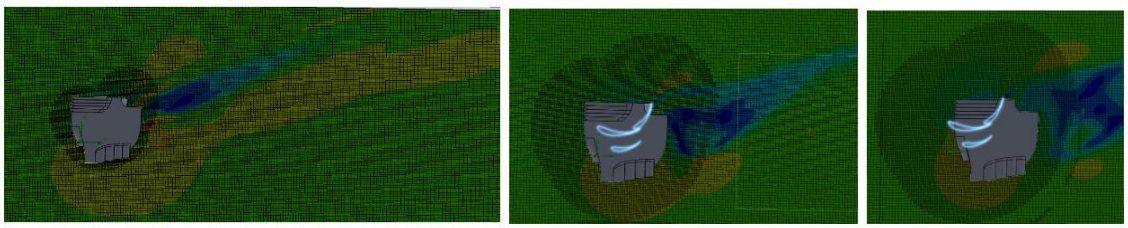

Fig. A.7 Rear wing v3 mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

Table A.9 Rear wing v4 mesh refinement configuration and results

MSHD with enplates v4-1

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	4341250	779	4:32:13	55,364	-119,86	5	1,2	1	0,67		
Mesh 2	4096593	999	5:23:48	56,285	-127,038	5	1,2	1	0,63		
Mesh 3	3393156	759	17:59:25	58,077	-105,511	5	1,2	1	0,5		
Mesh 4	3755917	1244	6:21:24	56,012	-122,925	5	1,2	1	0,57		

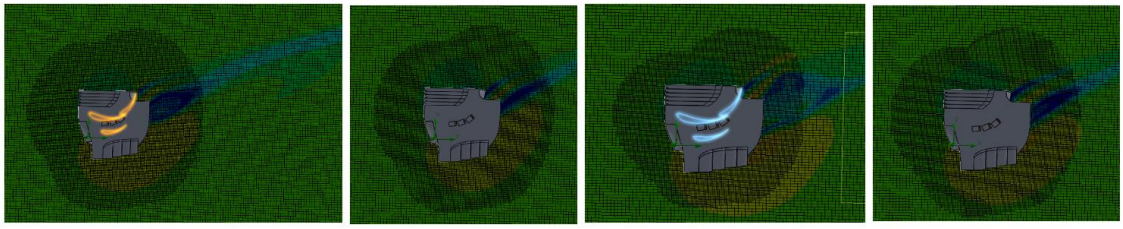

Fig. A.8 Rear wing v4 mesh refinement plots (mesh 1 –Left-, mesh 2 –center left-, mesh 3 –center right- and mesh 4 –right-)

Table A.10 Full car mesh refinement configuration and results

Full car

						Global mesh		Local Mesh			
	Total cells	Iterations	CPU Time	Fx	Fy	Lvl	Ratio Factor	lvl	shell 1	shell 2	shell 3
Mesh 1	5092337	343	2:48:06	118,074	-159,615	5	1,2	1	0,5		
Mesh 2	5809055	361	3:17:20	118,092	-159,595	5	1,2	1	0,6		
Mesh 3	6411377	373	3:44:02	118,009	-159,639	5	1,2	1	0,7		

Fig. A.9 Full car mesh refinement plots (mesh 1 –Left-, mesh 2 –center- and mesh 3 –right-)

APPENDIX B. LAP TIME SIMULATOR CODE

The files used in this project can be downloaded from the following github repository:

https://github.com/rubenbr99/LapTimeSimulator.git

APPENDIX C. LAP TIME SIMULATOR RESULTS

This appendix contains the results of the various simulations which graphs has been used along the document.

Table C.1 Fitness score evolution over generations test

generation	fitness score	track section of collision
1	113	
2	118	
3	228	first corner completed
4	245	
5	245	
6	246	
7	251	
8	254	midway through second corner
9	256	
10	256	
11	258	
12	350	start of the 4th corner
13	350	
14	351	
15	351	
16	351	
17	352	
18	352	
19	1000	2 full laps completed

Table C.2 Average lap time evolution over generations using speed mode method 1

Generation	Genome	Average Lap Time
1	0	37,11657
3	2	37,00054
4	1	33,9879
9	59	32,44357
9	89	32,37555
10	11	32,27953
11	50	32,28353
11	86	32,19151
11	87	32,06749
14	2	31,93546
18	30	31,87945
20	1	31,76342
24	5	31,39534
28	0	31,35133
28	2	31,35534
46	29	31,29532
48	0	31,29532
48	2	31,121531
90	1	31,2113
93	0	31,1953
100	24	31,2033

Table C.3 Average lap time evolution over generations using speed mode method 2

Generation	Genome	Average Lap Time
1	0	36,60846
1	1	35,65625
2	2	32,93167
6	5	32,77164
7	1	32,45157
8	1	32,22752
9	38	32,1475
9	49	32,07949
9	62	31,91145
10	32	31,77542
10	74	31,6594
12	65	31,61139
21	1	31,50337
25	1	31,44335
38	23	31,33133
40	26	31,2113
47	1	31,2033
48	70	31,15129
56	1	31,14729
75	1	31,08728
84	1	31,07928
84	92	31,04327
88	1	31,03527
92	1	31,02326
102	2	31,00326
109	1	30,96725
109	11	30,91924
111	7	30,85923
126	73	30,85523
126	78	30,79522
129	4	30,7352
130	0	30,7192
144	1	30,68319
156	0	30,67919
161	1	30,63518
183	6	30,59917
189	0	30,59517
222	0	30,57917

Table C.4 Average lap time evolution with a non-aerodynamic car

Generation	Genome	Average Lap Time
0	0	31,78343
5	1	31,76342
10	73	31,74342
12	16	31,70341
18	90	31,6754
21	1	31,6674
22	27	31,6434
22	37	31,63939
26	8	31,6354
30	18	31,63139
34	78	31,62739
37	88	31,62339
41	1	31,61939
50	11	31,61539
68	95	31,54737
72	76	31,52337
84	64	31,51937
89	52	31,48736
99	30	31,483336
152	0	31,483336

Table C.5 Average lap time evolution with aerodynamic car

Generation	Genome	Average Lap Time
0	0	30,86723
5	7	30,85923
6	60	30,84323
10	2	30,83922
12	19	30,83122
12	72	30,80322
21	62	30,79922
28	7	30,79522
29	52	30,75121
33	27	30,7232
34	1	30,69919
41	2	30,66319
43	22	30,65919
43	90	30,65519
45	89	30,64718
54	62	30,63918
57	24	30,63118
58	1	30,62318
62	4	30,61518
104	2	30,61118
140	84	30,60717
141	41	30,59517
150	73	30,59117
156	63	30,57517
164	0	30,57117
177	0	30,56717
218	0	30,56717