Politecnico di Torino

Politecnico di Torino

Automotive Engineering

A.Y. 2024/2025

Graduation Session October 2025

Improving Models and Simulations for Automotive Drag Predictions

Supervisors:

Jeff DEFOE

Luca MIRETTI

Candidate:

Philippe PESSION

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

CFD simulations for automotive bluff bodies often rely on simplified models for both the vehicle and the computational domain. Current models maintains simplified shape but fail in capturing the full flow behaviour over a real vehicle.

A model including vehicle body and tunnel geometry is presented in the literature and has been modified in order to improve its fidelity and reduce the differences with real life facilities. This model features an easy approach for the generation of the vehicle and wind tunnel models through 21 parameters as well as an automatic process for the setup of CFD cases, from meshing to simulation. Changes to the numerical schemes used by the CFD software are made to account for the variable quality of the grid. Moreover a moving ground system has been included to replicate the real system present in some automotive wind tunnels.

Another key aspect for the prediction of drag coefficient for automotive bluff bodies is that the same vehicle in different facilities often leads to different drag coefficients, with an average discrepancy of 6 to 10%. A parametric setup based on the improved models is carried out and a correction method is proposed by combining new sampling points with the results obtained previously in the literature. This new combined correction shows promising results while featuring a significanlty lower amount of samples, reducing the standard deviation by 43% compared to the current state of art when applied to a small dataset of 10 experimental points (5 for each vehicle in 2 different wind tunnels).

ACKNOWLEDGEMENTS

I want to thank my family for supporting me over these years, for believing in me and for giving me the chance to chase my dreams.

Thanks to the people I met in Canada and to my friends back home, you have been there to give me good memories during the study breaks.

Many thanks to Dr. Defoe for providing academic support for this work with his knowledge, allowing me to broaden my skills on this field of study. He provided amazing feedback and insights, improving the outcome of this work.

Finally, thank to Hannah for being so supportive to me, pushing me to be the best version of myself. You made me feel special and you filled my heart with a lot of memories that I will never forget.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
LHS	Latin Hypercubic Sampling
PCA	Principal Component Analysis
RBF	Radial Basis Function

1. Introduction

Aerodynamic resistance is also called *drag force* as it acts in opposite direction with respect to vehicle motion and it is defined as:

$$F_d = \frac{1}{2}\rho C_d A V^2 \tag{1.1}$$

where C_d is the *drag coefficient* and depends on multiple contributions:

- **pressure drag**: also called *form drag*, resulting from the pressure distribution perpendicular to the surface of the body;
- **skin friction drag**: resulting from viscous shear stresses over the contact surface due to the presence of the boundary layer.

Regarding automotive bodies, pressure drag is the leading effect with the skin friction drag contributing for around 10% to the total drag [1]. For this reason, optimization of body shape is of main importance during car design and can be used to reduce energy consumption.

In Figure 1.1, A is called *projection area* or *frontal area* and is defined as the area of the vehicle projected on a wall behind the body.

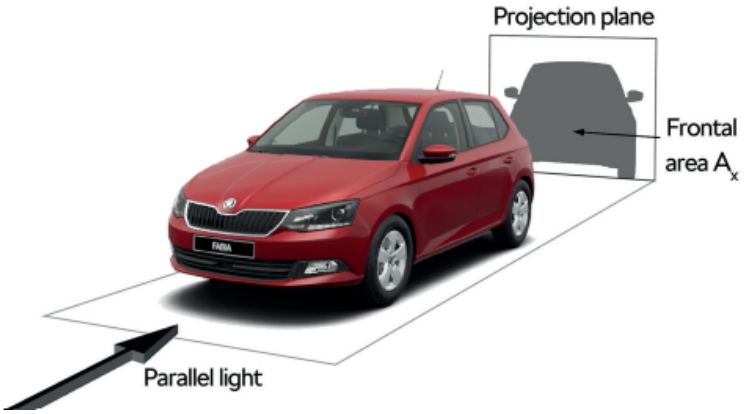


Fig. 1.1: Representation of frontal area for a vehicle [2]

ρ is the density of the fluid through which the object is moving and V is the speed of the fluid with respect to the vehicle.

Aerodynamics is becoming increasingly more important in the automotive industry due to its direct relationship with fuel/energy consumption and passenger comfort. Power required to overcome the aerodynamic resistance becomes the leading contribution to motion resistance at high speed as shown in Figure 1.2.

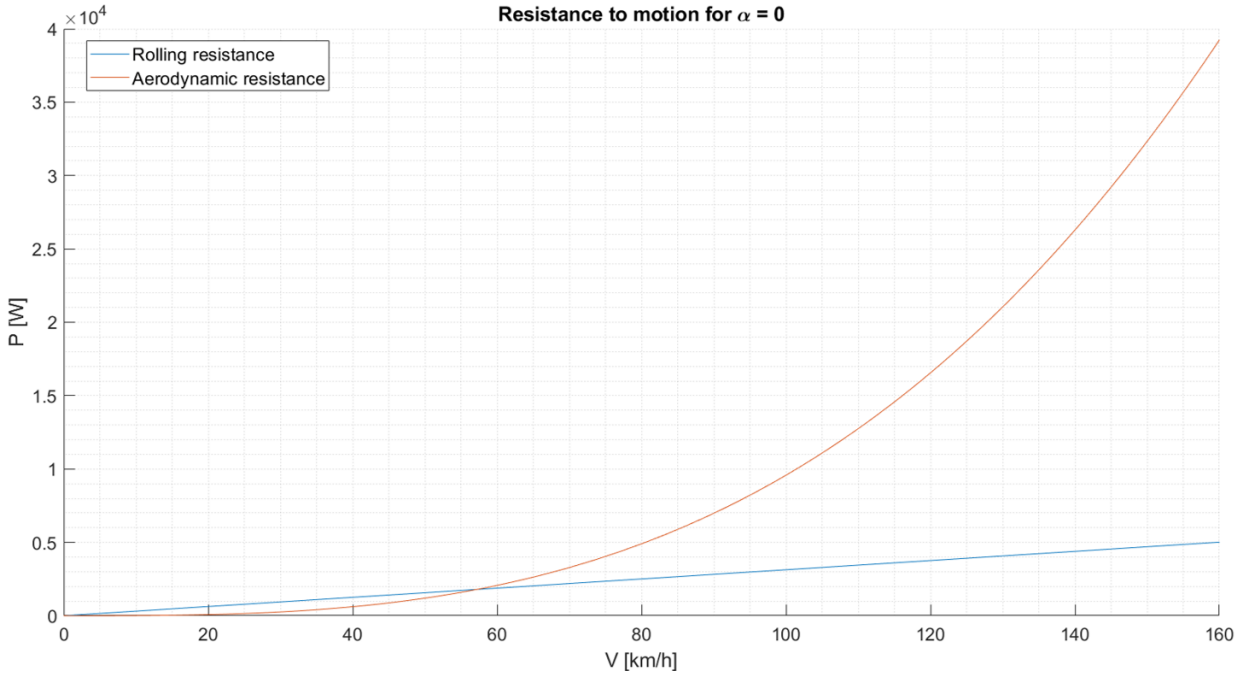


Fig. 1.2: Resistance to motion for a general vehicle traveling on a flat road

Accurate prediction of the drag coefficient is a key aspect for the design of a vehicle: C_d is used to assess fuel consumption as increasingly stricter regulations have forced the automotive industry to reduce vehicle emissions year by year. Moreover, with the introduction and increasing popularity of Battery Electric Vehicles, accurate prediction of drag coefficient becomes critical for range improvement of such vehicles.
Studies have shown that a 10% drag reduction can lead to a fuel consumption reduced by 5% at highway speeds, highlighting the importance of this quantity also for more traditional vehicles [3].

In the modern era, drag coefficient is measured by means of wind tunnel tests used to

replicate real road conditions by employing a fixed vehicle with air blown towards it. These tests are also used to validate design choices aimed at reducing the resistance to motion.
While useful, these type of tests require ad-hoc facilities and they are extremely time consuming as they require not only the setup of the vehicle within the test chamber but also the run-time of the test. Another problem is that to visualize certain aerodynamic quantities, specific transducers have to be used (for example pressure transducer to visualize pressure distribution on certain areas of the vehicle), requiring even more effort.
For these reasons, Computational Fluid Dynamics is used to have a first estimation of aerodynamic quantities prior to validate the results using the wind tunnel. This tool allows to setup a computational domain in which the full vehicle, or a part of it, can be simulated in a controlled environment in order to estimate drag coefficient or visualize flow behavior in an easier way.

1.1 Problem Statement

The biggest problem concerning wind tunnels tests is that the drag coefficient measured in a specific facility is highly dependent on its geometry: test performed using the same vehicle tested in different wind tunnels can lead to differences in drag coefficients ranging from 6 to 10% [4].
In addition, most of CFD simulations performed in the automotive industry rely on simplified domains, often failing in replicating the test geometries, effectively leading to different results when comparing real tests and simulations [5].

This study aims at improving the fidelity of simulation models, to reduce at minimum the differences between current wind tunnel facilities and computational domains. This improved approach is then used to retrieve a data-driven drag correction method which aim is to reduce the standard deviation of drag difference for the same vehicle in different wind tunnel facilities.
The performance of the correction method presented in this work is assessed on a small set of experimental data composed of drag values for 5 vehicles in 2 different wind tunnels but the approach presented can be applied also on datasets composed by an higher number of

vehicle and tunnel combinations.

1.2 Objectives

The work presented in this thesis has the objective of identifying the main sources of discrepancy between real tests and simulations models and addressing them by investigating the following objectives:

- Investigate the effect of changes in numerical schemes, improving the discretization approach used by the CFD software;
- Investigate the effect of the inclusion of a moving ground system in a wind tunnel domain;
- Define an improved data-driven correction approach, able to achieve similar results of the current state of art by reducing time requirements.

1.3 Overview of Contributions

This thesis proposes an automatic approach able to generate simulations models including features that allows to have computational domains as close as possible to real cases, while maintaining parametric variation for the generation of multiple different cases.
Moreover, an improved drag correction approach is presented in detail; this approach can be used as a baseline for future studies on the same topic.

1.4 Outline

Chapter 2 explains the current state of art of wind tunnel testing and CFD simulations, focusing on the models available in the literature.
The same chapter presents the current state of drag corrections, explaining why they are needed and for what effects they account for.

Chapter 3 focuses on the study cases performed, focusing on model generation and cases

setup with details about numerical schemes and the changes made to these latter. Implementation of the moving ground and derivation of the drag correction are also investigate in this chapter.

Chapter 4 presents the results of the work by investigating the effects of numerical schemes and implementation of the moving ground. Finally the results of the new correction approach are presented and compared to the current state of art.

2. Literature Review

Computing drag forces on objects from first principles requires solving the Navier-Stokes equations. Drag depends on pressure distribution and wall shear stresses [6]:

$$F_d = \int_A \left(-p\cos\vartheta + \tau_w \sin\vartheta\right) dA \tag{2.1}$$

where p is the pressure, τ_w is the wall shear stress and ϑ is the angle between the normal to the object surface and the freestream flow direction. As already pointed out in the previous chapter, in case of bluff bodies, pressure drag dominates on wall shear stress drag.

Experimentally the overall force can be measured, while numerical approach either require evaluation of local pressure and shear or the use of a control volume approach.

The aerodynamic loads that a model experiences during a wind tunnel test are measured through the usage of a wind tunnel balance (Julian et al. [7]). This device is composed by a single or multiple force transducers, capable of measuring forces and moments along several axes.

The main concern with wind tunnel testing is the high cost and time needed for wind tunnel experiments as presented by Ross et al. [8]. For this reason, CFD is an extremely important tool that can be applied in support of wind tunnel experiments to provide a complete description of external aerodynamics of an automotive bluff body. CFD employs a CAD representation of the vehicle model and a discretization of a fluid domain. The system of governing equations is solved iteratively for each cell in the computational domain, until a certain convergence criteria is met. An example of computational grid is given in Figure 2.1.

In the automotive industry good effort is put on the generation of the vehicle model, with this latter usually being a simplified version of a real production vehicle in order to minimize simulation time. The Ahmed body, presented by Ahmed et al. [9] and shown in Figure 2.2,

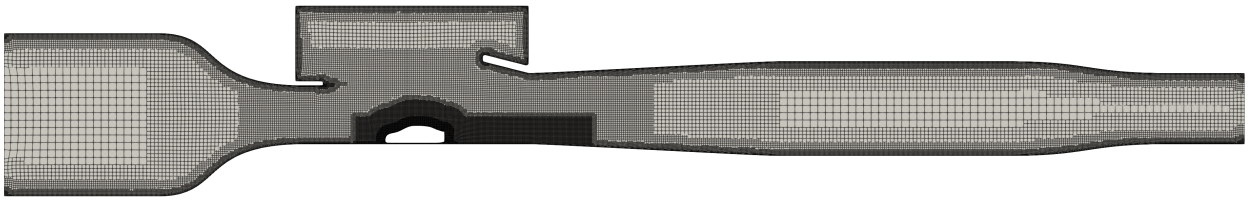


Fig. 2.1: Symmetry plane of a computational grid for CFD simulation

is one of the most common bodies used in the automotive literature but, while being simple, this model fails to accurately represent full flow behaviour around real production cars.

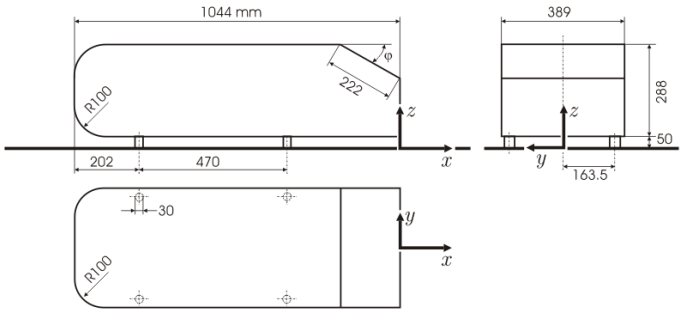


Fig. 2.2: Ahmed body [9]

An improvement is shown using the DrivAer model presented by Heft et al. [10] and shown in Figure 2.3. This model includes multiple features of real production vehicles, increasing the accuracy of results. Unfortunately this body has limited room for parameters variation, limiting the variety of geometries that can be simulated or tested.

Simplified vehicle models with parametric dimensional variability would thus be of use to qualitatively represent the complex geometry of a production vehicle while preserving a simplified shape for CFD simulations.

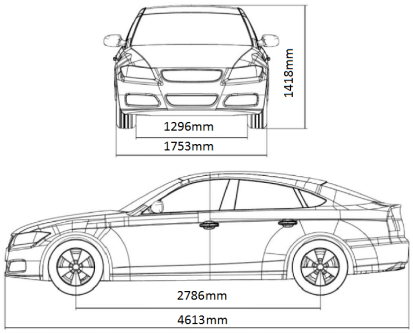


Fig. 2.3: DrivAer model [11]

Among all the possible CFD methods, *Reynolds averaged Navier-Stokes* (RANS) simulations are the most used in the automotive sector due to their reasonable accuracy and low computational cost. RANS employs an empirical turbulence model to allow for a reduced mesh size and critically, steady simulations. Bluff bodies involves large regions of separated flow and at high enough Reynolds numbers the separations are fundamentally unsteady. Ashton and Revell [12] found this method to be inaccurate in predicting the magnitude of aerodynamics quantities in such cases, requiring the need of more in deep studies on how to improve models to accurately predict aerodynamic quantities.

Additionally, Ljungskog et al. [5] have shown important improvements in drag predictions when the wind tunnel model is included in the virtual environment, showing good comparison with drag results measured in the actual wind tunnel prior to any correction. It has also been shown that, in comparison to a wind tunnel simulation, the base pressure for an open road domain is generally higher due to the finite nature of wind tunnels.

Fujs [4] developed a simplified model able to capture salient features of both vehicle body and tunnel geometry while retaining simplicity and parametric variation. This model is based on 21 parameters (12 for the tunnel and 9 for the vehicle), shown in Figure 2.4 and Figure 2.5. Only the tunnel and vehicle length are specified directly; all the other parameters are non-dimensionalised with respect to these two quantities. The number of parameters is minimized to have a compromise between simplicity and application scope.
The list of parameters with their description is presented in Appendix A.1.

A spoiler is added to the vehicles in order to enforce a physical point of separation.
Since the model is able to generate different vehicle types, an additional parameter is calculated to differentiate the generation of minivan geometry from the sedan model for which the spoiler location is different. t gives an estimate of the non-dimensional size of the trunk.

$$t = \frac{1}{l}\left(\frac{v}{w \cdot h} + h\right) \tag{2.2}$$

If $t > 0.62$ the minivan (or hatchback) geometry is generated, with 0.62 being the approximate change between sedan and hatchback.

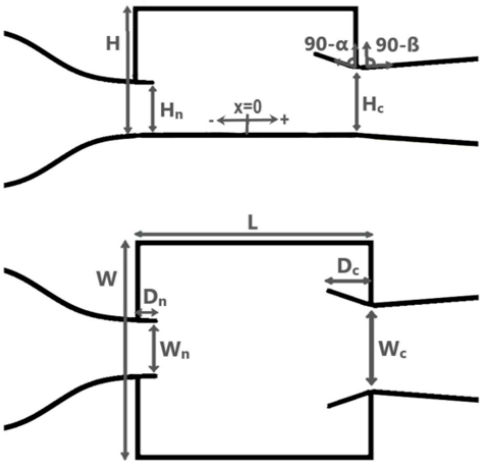


Fig. 2.4: Tunnel parameters

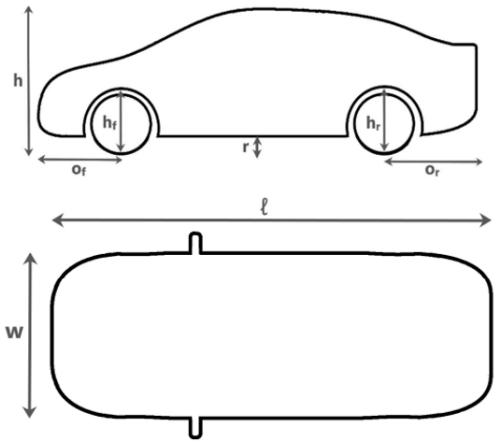


Fig. 2.5: Vehicle parameters

Only open-jet wind tunnels can be generated with this model. This type of tunnels is characterized by an open test section, as shown in Figure 2.6, which has the aim of reducing interference effects (Schuetz [13]). Wind tunnels are composed by 3 main sections:

- **nozzle**: this section has the aim of accelerating the flow and improve the quality of the flow at its outlet;
- **test section**: is the section in which the test object is placed;
- **collector**: this section has the aim of slowing down the flow and consequently recover pressure.

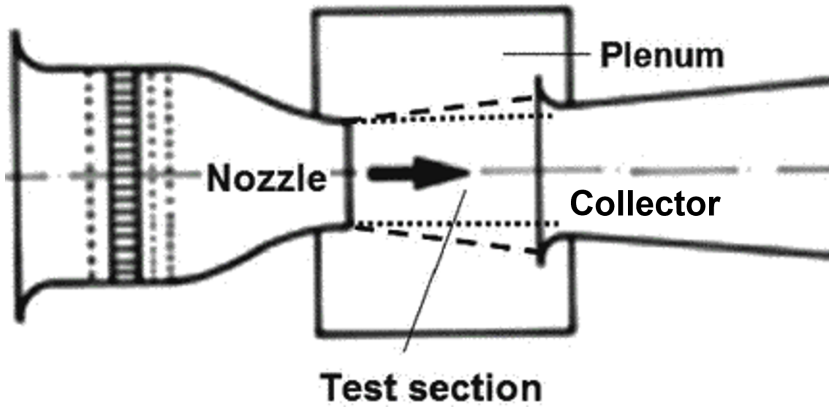


Fig. 2.6: Open-jet wind tunnel adapted from Schuetz [13]

The limited length of wind tunnels and the vehicle proximity to the tunnel walls create a discrepancy in the flow field compared to a real situation, resulting in different measured drag. The particular effects schematically shown in Figure 2.7 are listed here in detail:

- **jet expansion**: when a vehicle is present in the test section a jet of finite dimensions behaves differently to a jet of infinite extent;
- **jet deflection**: the limited distance between nozzle and vehicle can cause a deflection of the flow;
- **nozzle blockage**: the limited cross-section of the nozzle leads to differences in the flow field around the vehicle when compared to a free-stream case. Blockage ratio is defined as the ratio between frontal area of the test vehicle and nozzle cross-section area:

$$B = \frac{A_{\text{vehicle}}}{A_{\text{nozzle}}} \tag{2.3}$$

  As B increases, the effects on the test vehicle become more pronounced.
- **collector blockage**: interaction of the wake of the vehicle entering the collector;
- **horizontal buoyancy**: due to non-constant static pressure distribution within the test section. This effect leads to an additional drag force, called horizontal buoyancy force, that can be positive or negative.

Due to the presence of these effects, various correction methods are employed to reduce the discrepancies of drag coefficient for the same vehicle measured in different wind tunnels. Among all possible methods, the *classic correction* and the *two-measurements correction* are the most employed.

Classic correction, presented by Mercker et al. [14] includes a *dynamic pressure correction* dependent on 5 distinct velocity perturbations. Moreover, *horizontal buoyancy* is defined based on the frontal area and volume of the vehicle as well as the longitudinal rate of change of the static pressure coefficient.

Two-Measurements correction, presented by Mercker and Cooper [15], proposes a new approach to calculate the pressure gradient within the wind tunnel by means of an additional test with the vehicle in a perturbed pressure field (obtained through a physical blockage

behind the vehicle) with the revised correction that can be obtained by measuring the drag coefficient in both conditions. This method is generally adopted only for some reference dataset as it becomes impractical to double the measurements for each tested vehicle. Even if reference datasets demonstrate good relationships between corrected and measured drag coefficients (Lounsberry and Walter [16]), there is still a disparity in corrected drag coefficient for the same vehicle measured across different facilities; average difference across tunnels settles around 6%, but can go up to 10% for some vehicles.

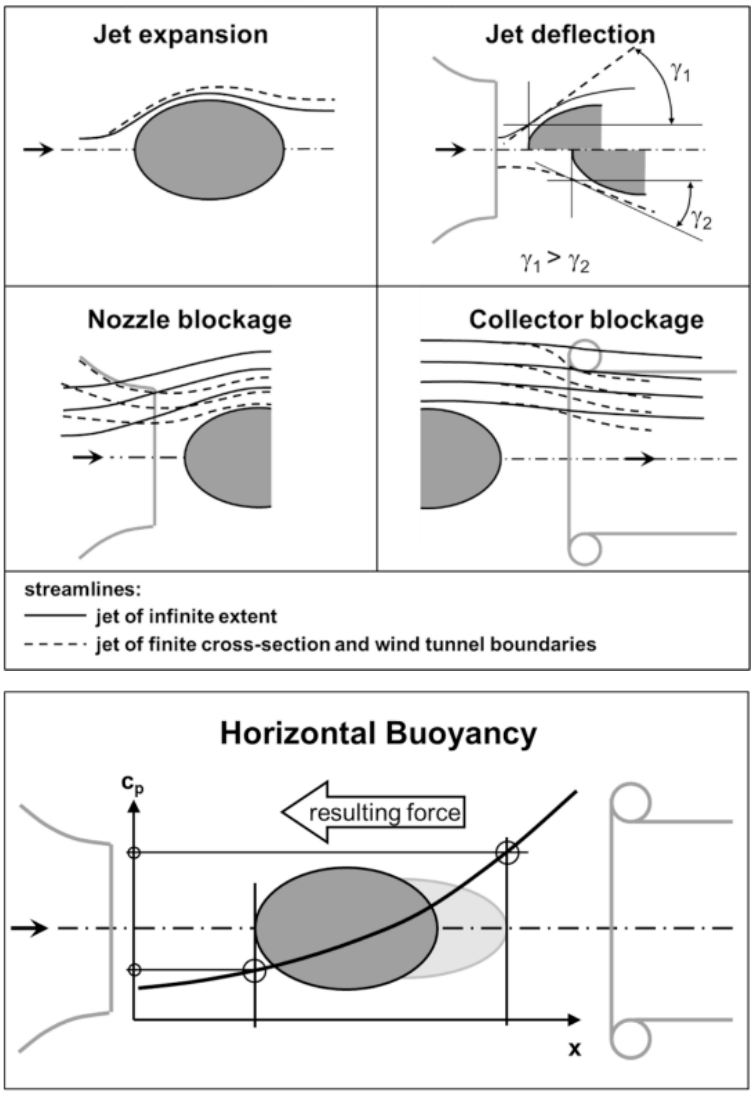


Fig. 2.7: Wind tunnel effects [17]

Fujs [4] developed a data-driven correction approach, aimed at reducing the dataspread of drag coefficient between different wind tunnels. This new correction method is based on

the difference between the drag coefficient in an open-road domain and the one measured in the wind tunnel, both quantified by means of CFD simulations.
The correction is derived upon the outcomes of a parametric analysis of several vehicle and tunnel combinations that frequently occur in industry.
A short-come of this approach is that simulations performed in the tunnel domain lacks the inclusion of a moving ground system and rotating wheels, effectively introducing some major discrepancies when compared to the real facility of Stellantis US, which has been upgraded with a 5 belts system, as can be seen in Figure 2.8.
Additionally, numerical schemes used for the discretization of quantities in the CFD software can be changed to improve the robustness of the simulations. More insights about this steps are given in Chapter 3.

Fig. 2.8: Mock-up vehicle in the upgraded wind tunnel of Stellantis US [18]

Although the number of parameters of the simplified model presented by Fujs [4] is a significant reduction compared to real geometry, it is still too high to pursue a direct parametric study. For this reason, Principal Component Analysis (PCA) presented by Bro and Smilde [19] is used to identify linear combinations of the parameters able to capture the maximum variance within the data.

To identify principal components, an experimental dataset was built using data of 5 different vehicle in 2 wind tunnels. PCA led to the result that the first 4 principal components

are able to capture 99.4% of the dataset variance and are thus the only ones considered. This PCs are only a linear combination of the input parameters; in particular PC_1 contains parameters related only to the wind tunnel geometry, while PC_2 through PC_4 are responsible for the vehicle geometry. This independence between tunnel and vehicle geometries in terms of principal components derives from the fact that only data from two tunnels was used for the analysis, suggesting that a broader dataset can lead to different coefficients values for all components. Reliance on the initial data though is not very significant as the process for the derivation of the correction method will remain the same in any case.

A *Latin Hybercubic Sampling* approach, presented by Loh [20], is used to retrieve new sampling geometries within the principal components space by using a stratified approach to increase input space coverage.
A total of 50 sampling points is generated and for each of them 3 simulations are carried out, namely in open road domain, in the wind tunnel domain and in an empty wind tunnel domain (used to retrieve pressure correction for the given tunnel geometry).
For each sample, the drag coefficient difference between the open domain and the wind tunnel domain, with pressure corrections applied, is calculated.

$$\Delta C_d = C_d^{open} - C_{d,q}^{tunnel} \tag{2.4}$$

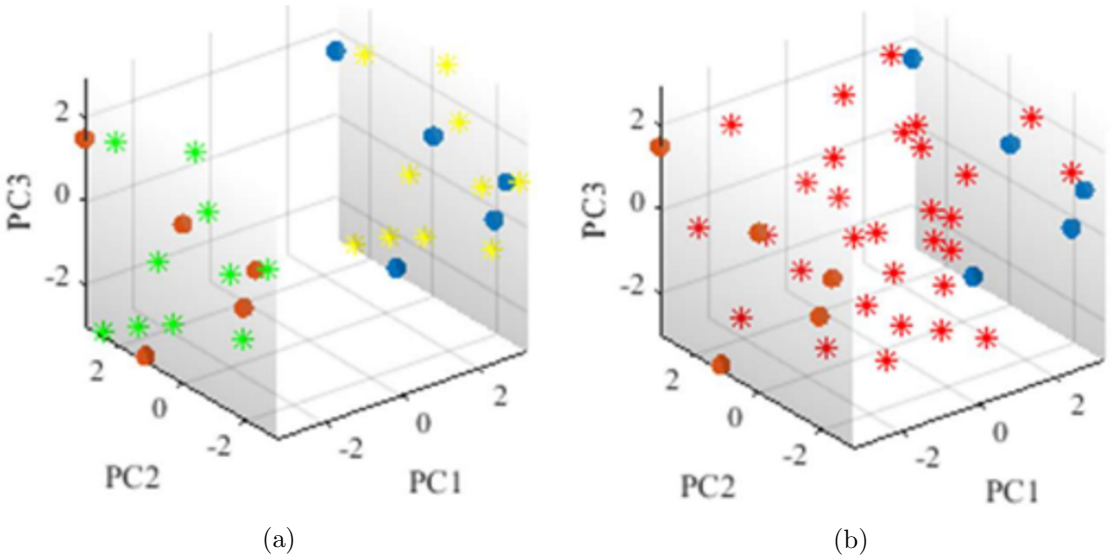
Radial Basis Functions (RBFs) are used to approximate functions of multiple variables; this type of function is used to fit scattered data and can be applied independently from the number of variables [21]. In our case, the values of ΔC_d from the CFD simulations on the sample geometries are interpolated using RBFs to create a 4D surface that returns the drag difference value based only on the four PC values of a particular geometry:

$$\Delta C_d = \mathrm{RBF}_n \left(PC_1, PC_2, PC_3, PC_4 \right) \tag{2.5}$$

where n denotes the number of sampling points used for the definition of the given RBF.
A set of 3 RBFs were defined, based on a different number of sampling points, as shown in Figure 2.9. In particular:

- **RBF$_{10+10}$**: based on sampling points taken by maintaining the same wind tunnel geometries (10 points for each tunnel in the initial dataset). This function is aimed at defining a correction tailored to the single wind tunnel geometries given the fact that open-jet facilities are limited.
- **RBF$_{30}$**: based on a reduced set of points and aimed at proposing a universal correction applicable to any vehicle and tunnel combination.
- **RBF$_{50}$**: based on the entire set of sampling points and aimed at showing the improvement of sampling density within PC space.

It is shown that RBF$_{50}$ is able to reduce the standard deviation by 59% compared the Two-Measurements method across the available experimental dataset.



(a) (b)

Fig. 2.9: Sampling points used by Fujs [4]. a) RBF$_{10+10}$, b) RBF$_{30}$

Studies on the inclusion of moving ground in the wind tunnel domain do not show clear results. In particular, Krajnović and Davidson [22] quantified a drag difference of 8% while including ground simulation on an Ahmed body. Moreover, they found the presence of two lower vortices arising close to the ground, which were not found with stationary ground simulations. On the other hand, Wang et al. [23] recently studied the effect of ground simulation on the estate-DrivAer model, finding a variation within 3% on drag coefficient results. The inclusion of ground motion showed significant differences only in proximity of

the ground, altering the surface pressure on the under-body of the vehicle. Due to this discrepancy, additional studies are required to quantify the effect of the inclusion of moving ground.

Table 2.1 shows a summary of the literature, showing the key findings and limitations for each research.

Table 2.1: Summary of researches

Authors	**Key findings**	**Limitations**
Ahmed et al. [9]	Most simple model in the literature	Lack in representation of full flow behaviour
Heft et al. [10]	Improved model that reduces difference to real vehicles	Limited parameters variation
Ashton and Revell [12]	RANS are useful to assess trends of aerodynamic quantities	RANS fails in predicting the absolute magnitude of such quantities
Mercker and Cooper [15]	Improved correction approach for the calculation of horizontal buoyancy	Increased amount of measurements for each test vehicle
Lounsberry and Walter [16]	TM approach improves the correlation between measured and corrected drag	Discrepancy in corrected drag coefficient for the same vehicle across different wind tunnels
Fujs [4]	Simplified model for both tunnel and vehicle with complete parametric variability	No moving ground for the tunnel and no rotating wheels for vehicle
Fujs [4]	Automated approach for setup of simulations	Numerical schemes do not account for possible poor quality of the grid
Fujs [4]	Improved performance of RBFs when used to reduce drag difference for the same vehicle in different wind tunnels	RBFs based on models which can be significantly improved

Previous studies highlights the evolution of models and drag corrections for simulations and test in the automotive industry. In general, automotive models presented in the litera-

ture fails in capturing full flow behaviour or they do not present full parametric variability. The approach presented by Fujs [4] provides good improvements compared to previous studies but, at the same time, introduces some more limitations that needs to be addressed in order to further improve the overall approach.

3. Methodology

The work presented in this thesis is a direct continuation of the project presented by Fujs [4]. In particular, the current work has the aim of improving the fidelity of models and simulations as well as updating the correction method presented previously to reduce the drag coefficient difference for the same vehicle in different wind tunnel facilities.

The previous work has shown really promising results suggesting that the overall process can be used systematically. The real advantage of the process presented is the fact that the setup of any case is fully automatic, from the model generation to the meshing of the computational domain, allowing to save time and resources.

3.1 Setup of the Cases

Cases presented previously include simulations of the vehicle in an open-road domain as well as in a wind-tunnel domain. While being completely different flow fields, the process for their setup is quite similar.

Vehicle and tunnel geometries are generated starting from 21 parameters, chosen as their values are publicly available for all passenger vehicles and tunnels. This choice enables the generation of all kind of geometries as the models do not require specific parameters known only to vehicle manufacturers.
The parameters are used to define 3D representation of the geometries using an open-source CAD software called SALOME. The advantage of using this software is the possibility of saving the generation process in a Python script which can then be modified accordingly in order to generate all kind of different models while only using the 21 parameters as input quantities.
In the model presented by Fujs [4], hatchback and sedan models are generated using slightly different processes due to different spoiler locations. The generation of the vehicle type is

determined using the trunk size parameter, with examples shown in Figure 3.1 and Figure 3.2.

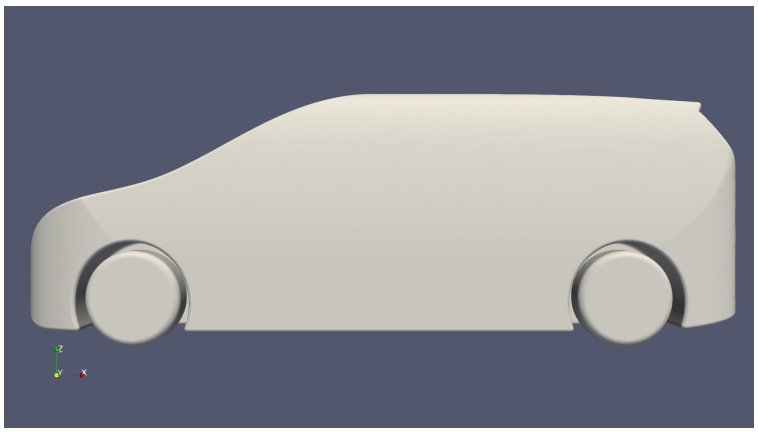


Fig. 3.1: Example of hatchback model (XZ view)

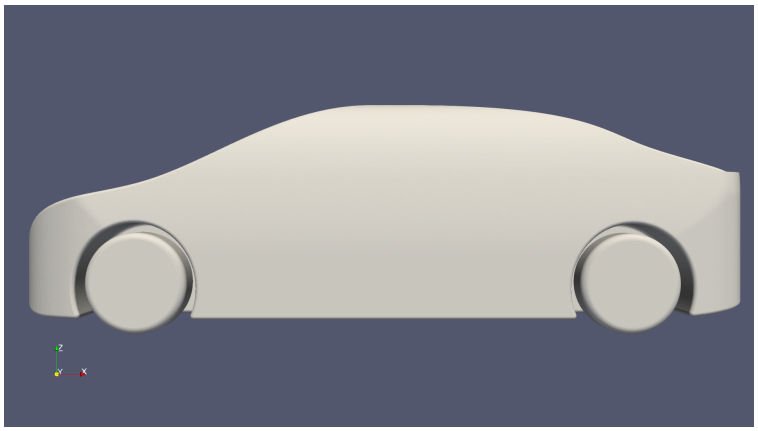


Fig. 3.2: Example of sedan model (XZ view)

Aerodynamic simulations are performed in OpenFOAM, a free open-source CFD software. This software allows for extensive customization of each case, from solver settings to numerical discretization schemes.

Reynolds averaged Navier-Stokes steady state simulation are performed as this type of approach is the most commonly used in the automotive industry for its ability in accurately predicting numerous flow characteristics. Reynolds decomposition and time-averaging are used on the set of Navier-Stokes equations to obtain RANS equations [24].

This approach relies on a complete approximation of turbulence by means of a set of transport equations. Multiple empirical models to analytically describe turbulence are available

in the literature but $k-\omega$ Shear Stress Transport model has shown better results compared to other turbulence models when performing RANS simulations on automotive bluff bodies [12]. This model is a blending of $k-\varepsilon$ in the far field and $k-\omega$ for cells near the wall [25], where k is the turbulent kinetic energy, ε is the turbulent dissipation rate and ω is the specific turbulent dissipation rate, defined starting from k and ε:

$$\omega = \frac{\varepsilon}{C_\mu k} \tag{3.1}$$

where C_μ is a constant equal to 0.09 [26].

To perform CFD simulations the domain has to be divided in a set of cells, called **mesh**, in which fluid dynamics equations are solved for each element. For the specific cases presented in this thesis the meshing process is composed of 2 steps:

- `blockMesh`: generates structured meshes starting from a dictionary file. This tool divides the domain in groups of one or more hexahedral blocks having straight or curved edges. The mesh is specified as a number of cells in each direction.
  In the cases studied in this work, this tool is used to generate a 3D block that encloses the desired domain. In case of tunnel simulations, the domain encloses the entire tunnel geometry while in case of open domain the enclosure generated by this tool coincides with the desired domain;
- `snappyHexMesh`: generates 3-dimensional meshes starting from triangulated-surface geometries (specified in STL format). A generic starting mesh is iteratively shaped to the object's surface. This tool needs a background mesh defining the boundaries of the domain (generated using blockMesh). This tool is used to carve the block domain to match the geometry of the tunnel but it is also used to refine the grid in some important regions for both tunnel and open cases.

Once the mesh is ready, the `simpleFoam` solver is used to perform a steady-state simulation for each case.

3.2 Boundary Conditions

Boundary conditions change depending on the domain to be simulated as patches for the tunnel domain are different compared to the ones used for the open-road domain. In this section, boundary conditions used for the previous study are presented, in order to have a clear picture of what is the state of simulations before making changes.

Flow conditions are presented in Table 3.1. It is important to notice that the inlet velocity for the wind tunnel domain differs from the free-stream velocity as there is the nozzle which accelerates the flow prior to entering the test chamber.

Table 3.1: Flow conditions [4]

Domain	$\mathbf{U}_\infty$ [m/s]	$\mathbf{U}_{\mathbf{inlet}}$ [m/s]	**I** [%]
Wind tunnel	30.56	5.53	3
Open road	30.56	30.56	0.1

Boundary conditions for the wind tunnel domain including the ones used for rotating wheels and moving ground system are available in Appendix B.1. Figure 3.3 shows the slip and no-slip surfaces for the tunnel domain.

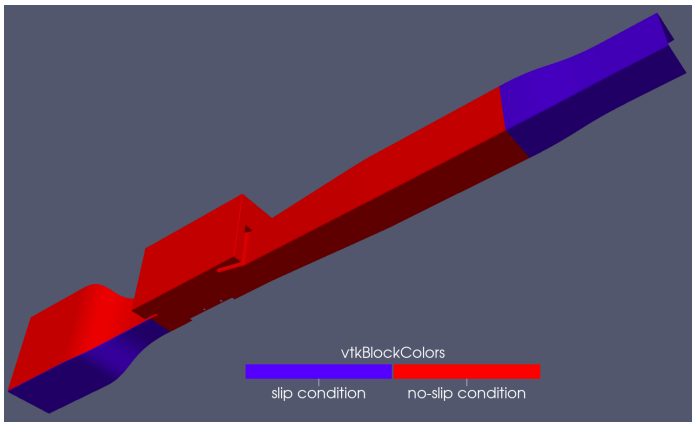


Fig. 3.3: Surface patches for wind tunnel

The ground patch in the open road domain has been set to have the same speed of the freestream flow in order to accurately represent a real situation.

Boundary conditions for each patch of the open domain are available in Appendix B.2.

3.3 Numerical Schemes

The set of partial differential equations known as governing equations, is defined for continuous fields and it thus must be discretized into a group of linear equations in order to work with the finite nature of the computational domain used in simulations.
The choice of numerical schemes affects how coefficients for these equations are calculated and consequently determines the characteristics of the solution.
Accuracy of the results depends on the quality and refinement of the computational grid: a grid independence study has been pursued by Fujs [4] and the refinement of the grid chosen previously has been used also in this case. Mesh properties can be used to assess grid quality and the most relevant for our case are:

- **skewness**: measures the relative distance between face centre and the line connecting neighboring cell centres (Figure 3.4)
- **non-orthogonality**: measures the angle between the surface normal to the face and the line connecting neighboring cell centres (Figure 3.5)

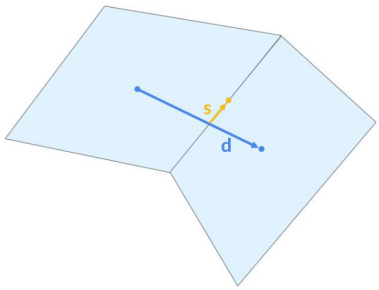


Fig. 3.4: Skewness [27]

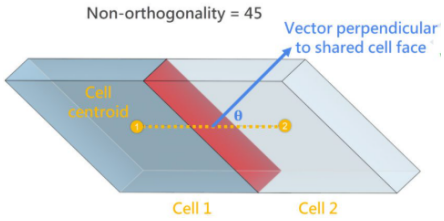


Fig. 3.5: Non-orthogonality [27]

The automatic meshing process compromises mesh quality. Because the mesh is generated using a uniform approach, without tailoring it to the specific geometry of the vehicle or tunnel, skewness and non-orthogonality can be significantly high with respect to ideal values. Due to the previously performed grid independence study, grid refinement is considered sufficient for the scope of this work. However, high values of skewness and non-orthogonality

translate in a poor mesh quality that will impacts numerical convergence and accuracy of the results. Numerical schemes can be changed to improve the robustness of CFD cases by maximizing the accuracy of simulations given the grid quality. An example of `checkMesh` output to get a sense of the grid quality is available in Appendix B.3

OpenFOAM uses a finite volume numerical method in order to solve the partial differential equations: instead of using a single control volume, the domain of interest is divided into connected finite volumes and the set of Navier-Stokes equations is then applied to each small volume, ensuring that mass and momentum fluxes across surfaces are consistent between the connected volumes. The discretization of the governing equations transforms equations for continuous fields into a system of linear equations applicable to discrete fields. The choice of numerical schemes influences the computation of coefficients for this discretization and determines the characteristics of the solution [28].

Discretization schemes are used to discretize each term in the governing equations: divergence schemes are numerical schemes used to discretize terms of the form $\nabla \cdot \boldsymbol{\Psi}$, where $\boldsymbol{\Psi}$ is a generic field quantity. In the CFD software, values for the quantities are stored at cell-centre so the key issue is the derivation of Ψ at face centre. In the work of Fujs [4], `divSchemes` were set to *1st order upwind* for both velocity and turbulence quantities (namely k and ω). This scheme represents the value at the cell face by the value of the upwind cell, as shown in Figure 3.6. Even though it can ensure boundedness of Ψ, pure upwind is highly diffusive due to the fact sharp gradients cannot form, which can lead to poor accuracy of the results [29].

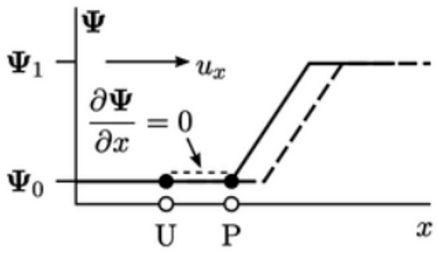


Fig. 3.6: Upwind scheme [29]

Numerical diffusivity arises from the truncation error introduced by first order approxi-

mation of pure upwind scheme: this effect results in a non-physical diffusion of a particular quantity, affecting the accuracy of the results. Low diffusion is desired to avoid capturing non-physical behaviour of the flow.

In order to improve the accuracy of the simulations given the grid quality, `divSchemes` for the velocity field have been changed to *linearUpwindV limited*. This scheme defines the face value as an extrapolation of the upwind cell value to the face, employing the upwind cell gradient, $\nabla\mathbf{\Psi}$, and a vector from the cell centre to the face centre, $\mathbf{d_P}$, as shown in Figure 3.8. This approximates a 2^{nd} order upwind approach on unstructured meshes.

linearUpwind schemes reduces the diffusivity of upwind and naturally corrects for skewness as it has a contribution in the direction between upwind cell centre and face centre [30].

The *V-scheme* is used to remove non-physical oscillations, by using the direction of the steepest gradient for vector quantities.

Furthermore, *limited* entry is employed to ensure that if the face value exceeds the boundary of the neighboring cell centre, the gradient is modified to match the bounding value [31].

The increase in accuracy for the *linearUpwind* scheme comes with a reduction in numerical stability due to oscillations possibly introduced by higher order schemes. Total Variation Diminishing (TVD) schemes are used as a compromise of boundedness and accuracy where a limiter β is used to calculate the value at the face:

$$\Psi_f = (1 - \beta)\,\Psi_U + \beta\Psi_L \tag{3.2}$$

where Ψ_U and Ψ_L are the value of the quantity from upwind interpolation and linear interpolation, respectively [32]. β factor is calculated based on the change of gradient of Ψ between face and upwind cell. For the simulations performed in this thesis, linearUpwind scheme applied to turbulence quantities lead to divergence of the solution so the *Minmod* scheme is used instead. This option is a TVD scheme which limits to pure upwind, as shown in Figure 3.7. It is more diffusive than *linearUpwind* but still more accurate than the pure upwind scheme used previously [33].

The `fvSchemes` file for numerical schemes is available in Appendix B.4.

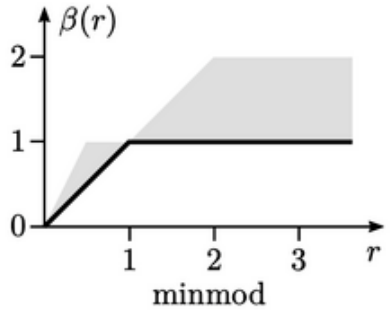


Fig. 3.7: Minmod scheme [32]

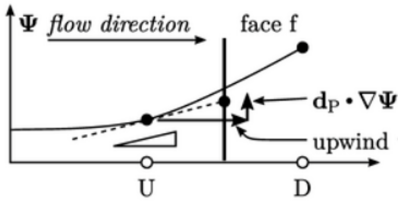


Fig. 3.8: linearUpwind scheme [30]

3.4 Convergence and Performance Metrics

The aerodynamics of a vehicle is highly complex due to the three-dimensional flow and extensive turbulent wake zones, generally defined by longitudinal rotating vortices [9].
In this study, steady state simulations are performed on a fundamentally unsteady problem. Using RANS simulations, turbulence is simulated by means of empirical models, which are not able to fully capture the real behaviour of the flow, especially in the wake where turbulence is significant thus affecting drag results. The choice of steady simulations leads to drag values that are oscillating as the number of iterations increase, without reaching a single convergent value.
Since drag coefficient is of primary interest for this study, we need to define an approach to assess whether the simulation can be considered converged. In this case convergence of the cases is of major interest in order to be able to compare different simulations: the same criteria used for each case will ensure that results are comparable to each other. In addition, we need to determine a single value for the drag coefficient, which will then be used to obtain the correction based on the discussion in the previous chapter.

Experimental uncertainty on drag coefficient measured in real wind tunnels is approximately ± 0.002 [34]: considering that modern passenger cars have C_d between 0.2 and 0.3, this uncertainty ranges around 1%. The convergence criteria chosen for this study is coherent with the experimental uncertainty: the minimum to maximum difference of $C_{d,mean}$ has to be within 1% of its value for the simulation to be considered converged. An additional metric is defined to ensure that the local behaviour of $C_{d,mean}$ remains stable for a

sufficiently high number of iterations. More insights about these criteria are given in the following paragraphs.

Drag coefficient is averaged over a 10,000 iteration moving window. The `movmean` function on MATLAB has been used to perform this action: when there are insufficient elements to fill the window, the size is automatically chopped at the endpoints and the average is taken over just the items that populates the window. For this reason, the first 10,000 iterations are only used to populate the function with the right amount of values, and it is important for the convergence point to not fall in within this "transient" region, where the behaviour of the mean drag coefficient can be stable but its value can be inaccurate.

To assess the convergence point, multiple metrics have been defined. First, a normalized drag variation coefficient is defined as the ratio between the minimum-to-maximum difference of the mean drag coefficient over a 10,000 iteration window and the mean drag coefficient averaged on the same number of iterations:

$$C^*_{d,mean} = \frac{\Delta_{min-to-max}(C_{d,mean})}{C_{d,mean}} \tag{3.3}$$

This metric is not sufficient to assess convergence as its absolute value does not give any insight on $C^*_{d,mean}$ local behaviour as the iterations increase. For this reason, the minimum-to-maximum difference over a smaller iteration window of 3,000 iterations is computed:

$$\Delta C^*_{d,mean} = \Delta_{min-to-max}(C^{*,\max}_{d,mean} - C^{*,\min}_{d,mean}) \tag{3.4}$$

For the simulation to be considered converged, two conditions on these metrics have to be satisfied. The value for $\Delta C^*_{d,mean}$ has been chosen based on the minimum value among all simulations and the same limits have been used for all cases in order to have comparable results.

$$\begin{cases} C^*_{d,mean} < 0.01 \\ \Delta C^*_{d,mean} < 1 \cdot 10^{-4} \end{cases} \tag{3.5}$$

From Figure 3.9 we can better understand how this double condition works.

The first metric ensures that the convergence point does not fall into the first 10,000 iteration span, where the MATLAB function is calculating the mean drag coefficient with a reduced amount of iterations. The second metric then ensures that the behaviour of $C^*_{d,mean}$ remains stable for a sufficient amount of iterations, meaning that the simulation has reached a local convergence where the drag value remains stable.

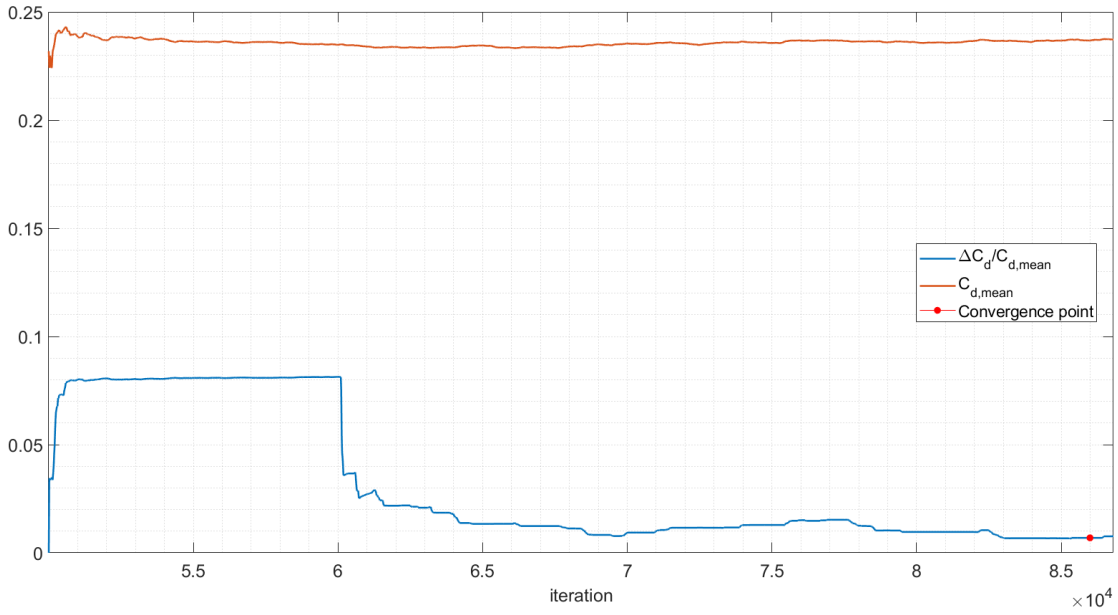


Fig. 3.9: Example of convergence for a generic case

For this specific case, it is important to notice that the simulation with the improved numerical schemes started from the 50,000$^{\text{th}}$ iteration as this was the ending point of the previous simulations carried out by Fujs [4].

A similar approach is used to assess the convergence of simulations performed in the empty wind tunnel.
In this case drag coefficient cannot be used as no vehicle is present in the test chamber and pressure is used instead. Equation 3.6 follows the same approach used for convergence assessment of drag, with the only difference that in this case the normalization is now done on the dynamic pressure at nozzle exit.

$$p^*_{mean} = \frac{\Delta_{min-to-max}\left(p_{mean}\right)}{\left(p_0 + {}^1\!/_2\rho U^2\right)_N} \tag{3.6}$$

where p_{mean} is calculated at the location where the front of the vehicle lies when it is placed in the tunnel and p_0 is the value of static pressure taken at nozzle exit. The value of U used to calculate the dynamic pressure is taken at the same location of p_0.

Equation 3.7 also uses the same approach used for drag, where the quantity is calculated on a smaller iterations window of 3,000 iterations.

$$\Delta p^*_{mean} = \Delta_{min-to-max} \left(p^{*,\max}_{mean} - p^{*,\min}_{mean} \right) \tag{3.7}$$

Pressure has higher absolute values compared to drag, so conditions for the convergence are more relaxed, as shown in Equation 3.8. It is important to point out that in this case, the check is performed on the absolute value of p^*_{mean} and Δp^*_{mean} as a consequence of the pressure being able to have negative values.

$$\begin{cases} |p^*_{mean}| < 1 \\ |\Delta p^*_{mean}| < 0.5 \end{cases} \tag{3.8}$$

3.5 Inclusion of 5 Belts Moving Ground System

To further reduce differences between the wind tunnel domain and real life environment, a moving ground system can be used to replicate road movement underneath the vehicle. This system is usually implemented by means of 5 belts: four Wheel Driving Units (or WDUs) under each of the 4 wheels and one center belt placed below the under-body of the vehicle. A schematic view of the system is shown in Figure 3.10.

Fig. 3.10: Schematic top view of 5 belts system

Studies pursued in the literature do not show a particular trend for the inclusion of the moving ground even if this system attempts in reducing differences between CFD and real road cases.
In this case, the implementation in OpenFOAM doesn't require major effort and so there was no reason why the system shouldn't be implemented. Technical drawings of the system used by Stellantis US have been provided in order to ensure a correct implementation in the computational domain.

Computational domains used in this study aim at simulating only half of the vehicle, with a symmetry boundary condition applied at the cutting plane, thus only half of the 5 belts system has to be implemented.
OpenFOAM offers a very useful function, called `searchablePlate`, that allows to implement two dimensional plates in the computational domain. The plate is defined through two quantities:

- **origin**: specifies the corner of the plate;
- **span**: specifies the dimensions in the 3 directions, where one of them has to be a 0 entry.

For example, for a plate defined on the xy plane:

$$\begin{cases} origin = (O_x,\, O_y,\, O_z) \\ span = (S_x,\, S_y, 0) \end{cases} \tag{3.9}$$

The plate will be from $(O_x,\, O_y,\, O_z)$ to $(O_x + S_x,\, O_y + S_y,\, O_z)$.
This function has to be implemented in the `snappyHexMeshDict` file, where settings for *snappyHexMesh* are defined.
Based on the dimensions given by our industrial partner, a mathematical generalization can be retrieved in order to adjust the position of the system to accommodate different types of vehicles. Dimensions of the system are also generalized in order to maintain the same proportions of the real system independently from the length of the test chamber.
Equations for the **origin** are shown in Table 3.2, where:

- $\mathbf{x}_{shift}$: represents a shift of the system towards the nozzle exit plane (upstream), calculated based on tunnel length:

$$x_{shift} = \frac{0.738}{L_{AAWT}} \cdot L_{genericTunnel} \tag{3.10}$$

where 0.738 is the shift of the system in the AAWT tunnel, L_{AAWT} is the length of AAWT test section and $L_{genericTunnel}$ is the length of the test section for any generic tunnel configuration;

- $\mathbf{l}_{genericBelt}$: represents the length of the considered belt based on tunnel length.

Table 3.2: Equations for the origin of belts for moving ground system

searchablePlate	**x**	**y**
centerBelt	$-x - \frac{l_{centerBelt}}{2} - x_{shift}$	0
frontLeftBelt	$-x - l\left(\frac{1}{2} - \frac{o_f}{l}\right) - \frac{l_{FrontLeftBelt}}{2}$	$\frac{w}{3}$
rearLeftBelt	$-x + l\left(\frac{1}{2} - \frac{o_r}{l}\right) - \frac{l_{rearLeftBelt}}{2}$	$\frac{w}{3}$

Span of each plate is defined by using the dimensions calculated based on tunnel length according to Equation 3.11 where j represents a general parameter of the system, L_{AAWT} is the test section length of the wind tunnel used by Stellantis US and L_{tunnel} is the length of the test section for any tunnel configuration.

$$j = \frac{j^{\text{AAWT}}}{L_{\text{AAWT}}} \cdot L_{\text{tunnel}} \tag{3.11}$$

For example, for the center belt:

$$l_{centerBelt} = \frac{l^{\text{AAWT}}_{centerBelt}}{L_{\text{AAWT}}} \cdot L_{\text{tunnel}} = 0.416 \cdot L_{\text{tunnel}} \tag{3.12}$$

where, by substituting any value of L_{tunnel}, the span of the center belt for that specific tunnel

configuration can be retrieved.

In order to fully implement the moving ground system, changes to the vehicle model had to be carried out. Up to this point the CAD representation of the vehicle was a single model, including body and wheels. This approach allows to assign a single boundary condition thus cannot accomodate the implementation of rotating wheels.
Wheels need to be split from the body and they have to be exported in different files that will then be defined in the computational domain using different boundary conditions compared to the vehicle body. Since the 3D model is entirely defined on SALOME, this step can be done interactively on the software and saved to a new Python script to facilitate the generation of future models.
Moreover, a region emulating the contact patch of the wheel has been removed as the meshing process already creates that region when merging the wheel patch to the WDU. The updated Python code used for the generation of the improved models is available in Appendix B.5
Visualization of the different patches is shown in Figure 3.11.

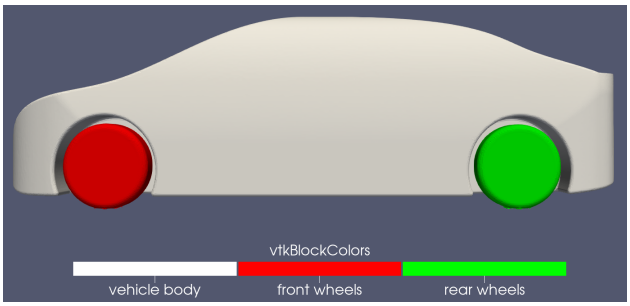


Fig. 3.11: Highlights of different patches for vehicle body and wheels (mirrored side view)

Rotating wheels are implemented through `rotatingWallVelocity`, an OpenFOAM boundary condition for the velocity field. This condition is defined by *origin*, *axis* and *rotational speed*.
Mathematical relationships are retrieved in order to accommodate different types of vehicles, as done for the 5 belts system; in this way the whole process can be generalized and automated to facilitate the creation of new study cases.
Equations for the origin of rotating wheels patches are shown in Table 3.3.

Table 3.3: Equations for the origin of rotating wheels

Patch	**x**	**y**	**z**
frontWheels	$-x - \frac{l}{2} + o_f$	$\frac{w}{2}$	$\frac{5}{12}h_f + 0.066l$
rearWheels	$-x + \frac{l}{2} - o_r$	$\frac{w}{2}$	$\frac{5}{12}h_f + 0.066l$

Rotation occurs around the $-y$ axis, where the negative value is needed to have the rotation in the right direction. The axis of rotation remains the same for all configurations as all of them share the same orientation of the axes.

Wheels are considered rigid bodies so rotational speed is set through:

$$\omega = \frac{U_\infty}{r_{\text{wheels}}} \tag{3.13}$$

where $r_{\text{wheels}} = \frac{5}{12}h_f$.

The results of this implementation are shown in Figure 3.12 and Figure 3.13 for a generic configuration.

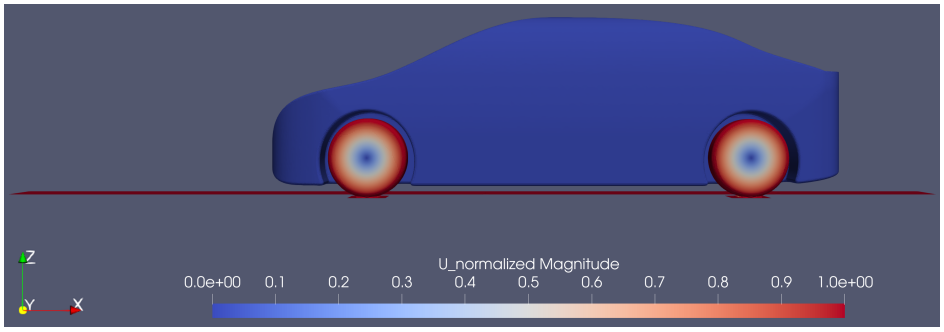


Fig. 3.12: $\frac{U}{U_\infty}$ for a generic configuration (mirrored side view)

Figure 3.12 shows clearly how the rotation is implemented: the velocity of the wheel grows from the center toward the outside, with the external part having the same speed as the moving ground.

Figure 3.13 highlights the direction of rotation of each wheel, by showing the magnitude of

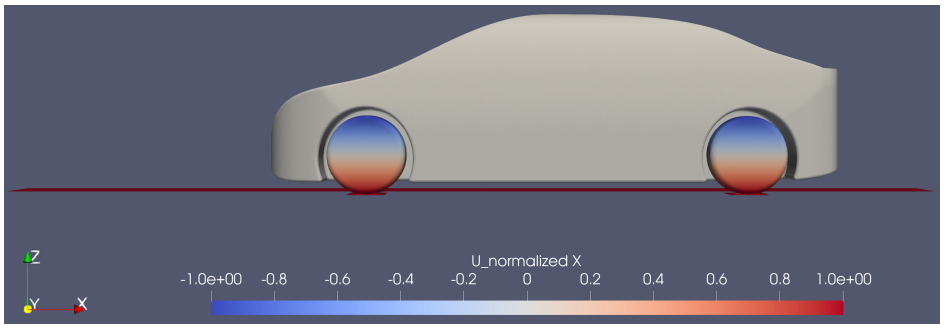


Fig. 3.13: $\frac{U_x}{U_\infty}$ for a generic configuration (mirrored side view)

the velocity in the longitudinal direction: the bottom half of the wheel and the belts shares the same magnitude and direction of the normalized velocity while the top half has the same value but with opposite direction, thus showing how the wheels are rotating correctly compared to the direction of the flow.

3.6 Setup of Parametric Study

The aim of this study is the definition of a new correction method based on the improved models discussed up to now. The approach previously used by Fujs [4] has been found to be particularly good in reducing the standard deviation of the difference in drag for the same vehicle in different wind tunnels and for this reason the same procedure has been used.

Due to time limitations, the amount of sample geometries used previously cannot be replicated, thus a correction based on a reduced amount of sampling points has to be carried out.
Figure 3.14 demonstrates how RBF_{50} behaves in the principal component space. A grid of two-dimensional plots with axes PC_3 and PC_4 at each specific (PC_1, PC_2) value can be seen. It can be seen that the function doesn't highlight any sudden variation in the space, with gradual variations in all directions. This detail suggests that a smaller set of sampling points can still capture important information within the used dataset, and can thus be employed to retrieve a good correction method.

The same experimental dataset used previously, based on data from 5 vehicles in 2 dif-

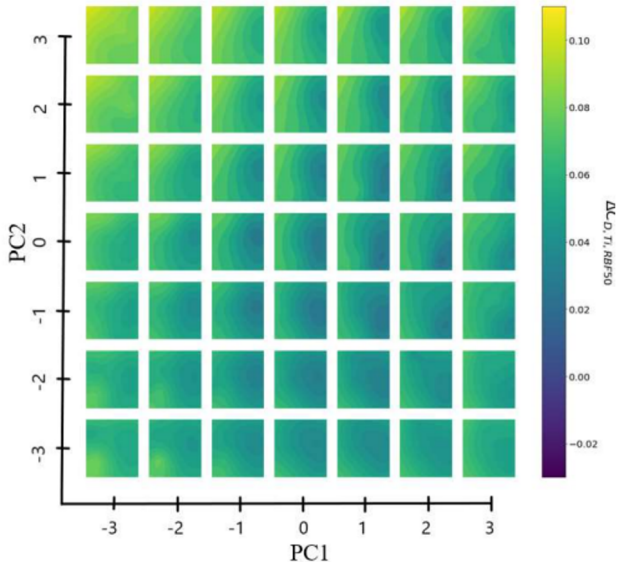


Fig. 3.14: RBF_{50} behaviour in the principal component space from Fujs [4]

ferent wind tunnel facilites, has been rebuilt in MATLAB. *Principal Component Analysis* is performed again as the coefficients of PC equations listed in the previous thesis have been found to be inaccurate, leading to unexpected results in the next steps. New equations to compute the PCs are listed in Appendix B.6.

In order to have the data as close as possible as the one used before, a Z-score normalization of the dataset is performed prior to PCA, using MATLAB `zscore` function [35]. This method, presented in Equation 3.14, transforms the data to have zero mean and unity standard deviation, so that each parameter has the same scale.

$$x^* = \frac{x - \overline{x}}{x_\sigma} \tag{3.14}$$

where x is a generic parameter, $\overline{x}$ is the mean value of parameters in the dataset and x_σ refers to the standard deviation.

Coefficients found for PC_2 to PC_4 are slightly different to the ones found previously but again the first 4 Principal Components have been found to be able to capture 99.4% of the dataset variance, as shown in Figure 3.15, thus will be the ones considered from now on.

Latin Hypercubic Sampling has been performed on the experimental dataset in order to identify new sampling points in the PC component space. Coefficients found from PCA are

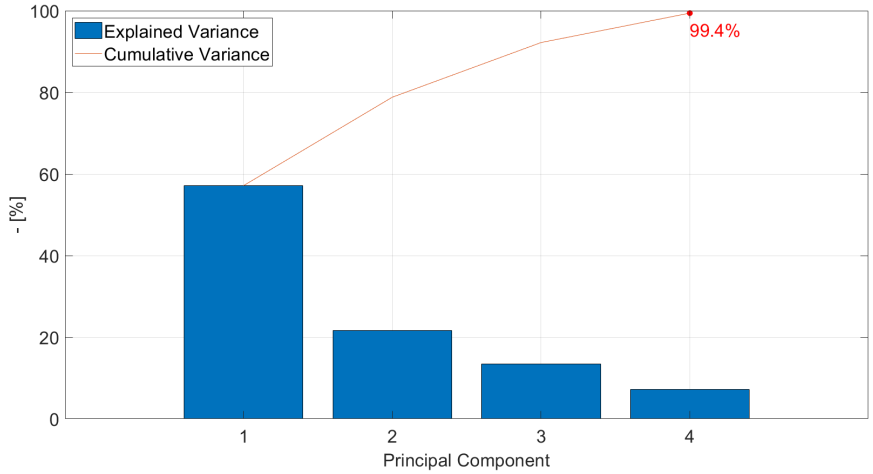


Fig. 3.15: Dataset variance

used on the initial dataset to retrieve a set of PC values for each experimental geometry: a combination of a certain vehicle in a particular wind tunnel will lead to a set of 4 values, one for each PC. These PC values are then plotted on the same figure as the sampling points to have a comparison between the values of the experimental dataset and the values of new geometries.

`lhsdesign` function on MATLAB [36] has been used to perform the sampling step: the function requires the number of sampling points and the ranges of PCs of the reference dataset. The function returns values from 0 to 1 so the outcome of the function has to be scaled to match the range of PCs of the experimental dataset.

A set of 10 sampling points is generated and all of them have been found to be well spaced within the PC space, as shown in Figure 3.16 where the red and green points represent experimental data for the 5 vehicles in the first and second wind tunnel, respectively while the yellow points are the new samples defined by LHS. PC values for each sampling point are listed in Appendix B.7.

The next step is translating the values of LHS points from PC space to parameters space. In particular, an under-determined problem has to be faced as only 4 equations are available (one for each Principal Component) but 21 parameters have to be estimated.

In order to solve this issue, an optimization algorithm aimed at minimizing a certain metric can be used. `lsqnonlin` on MATLAB has been used [37]; this function solves non-linear

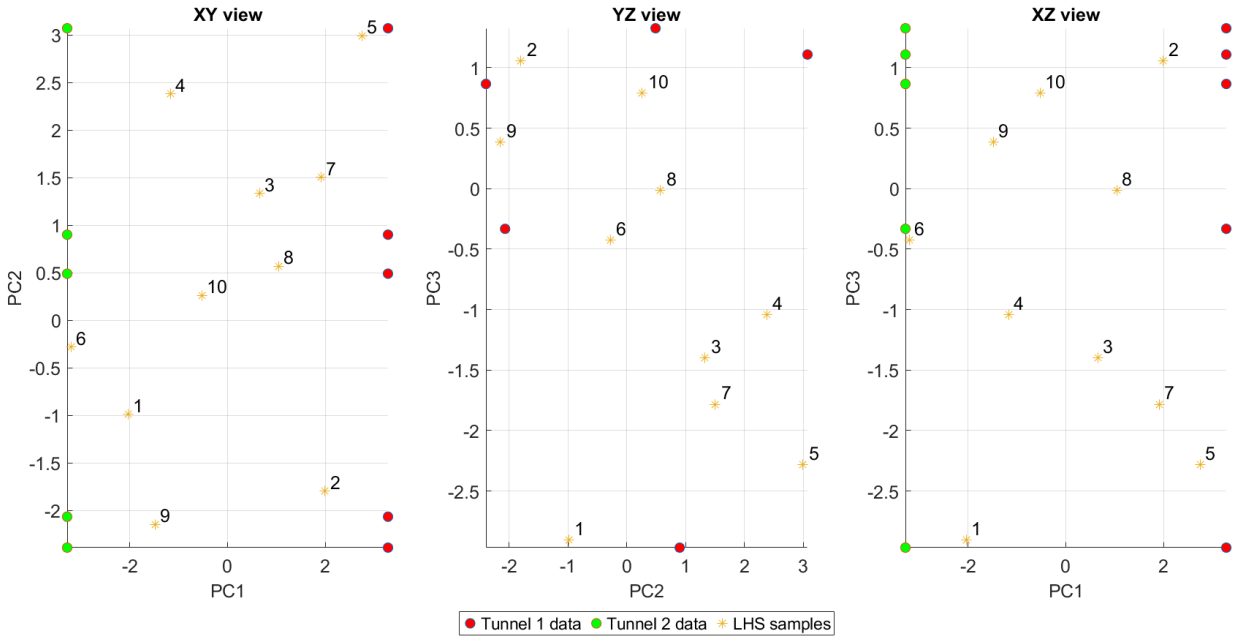


Fig. 3.16: LHS sampling points in the PC space

least-squares problems and requires the definition of metric to be minimized in vector-valued form. The function internally performs the sum of squares for each vector value.

The quantity used for the algorithm is the difference between the PC values of a single sampling point and the PC values calculated with the estimated parameters:

$$J = \mathrm{PC}_{\mathrm{LHS}} - \mathrm{PC}_{\mathrm{estimated}} \tag{3.15}$$

The optimization function also requires lower and upper boundaries for each parameter. Ranges have been taken from the previous work and have been tweaked in order to accommodate some limitations of the geometry generation process: stricter ranges have been defined in order to avoid errors when model were generated through SALOME with the improved Python code.

In addition, an initial guess is required by the MATLAB function: the algorithm starts from the initial guess and finds a minimum of the sum of squares of the objective function J. Since the mathematical problem is underdetermined, an infinite amount of set for the geometry parameters can be found for a specific set of PC values. By keeping the same initial guess for all the sampling points, very similar set of parameters are given by the optimization

algorithm.

A different initial guess for each sample leads to different sets of parameters and consequently to different geometries. This step is achieved by choosing the initial value of each parameter to be a random value between the lower and upper boundaries. To have consistent results every time the code is executed, MATLAB's random number generator is locked to the default value.

Complete MATLAB code used for this step is listed in Appendix B.8.

Once parameters are retrieved for each sampling point, geometries for each of them can be generated using the improved Python script developed as described in the previous chapter. For each configuration 3 different simulations have to be performed in 3 different domains: open road, wind tunnel and empty wind tunnel, with this latter performed in order to retrieve dynamic pressure correction and horizontal buoyancy correction. Open-road domain is characterized by the ground patch having a `movingWallVelocity` boundary condition, in order to replicate as close as possible a real situation.

Drag coefficient is calculated through `forces` function object available in OpenFOAM [38]. Frontal area is needed by the function in order to calculate drag coefficient for each configuration. Its value is quantified using `pArea` [39] package on Python starting from the STL model.

Table 3.4 shows the average cell count for each of the simulated domain.

Table 3.4: Cell count for computational domains

Domain	**Number of cells** $[\cdot \mathbf{10^6}]$
Tunnel	15
Empty tunnel	10
Open	40

Sample images for all configurations in the wind tunnel domains are shown from Figure 3.17 to Figure 3.26. It is important to notice that geometries differs not only for the different types of vehicles but also for different wind tunnel geometries and positions of the

vehicle within the test chamber.

In some of these configurations the vehicle is placed in close proximity of collector entry; while not being realistic configurations these cases are useful to check the performance of pressure corrections.

Fig. 3.17: configuration 1 - Tunnel domain

Fig. 3.18: configuration 2 - Tunnel domain

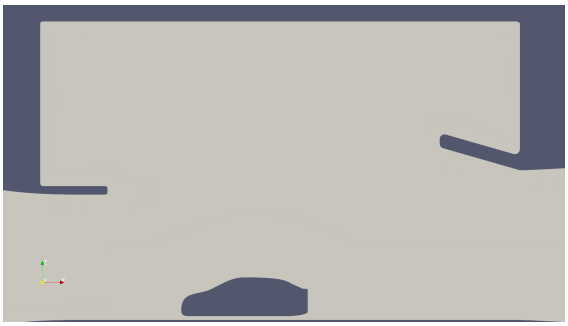


Fig. 3.19: configuration 3 - Tunnel domain

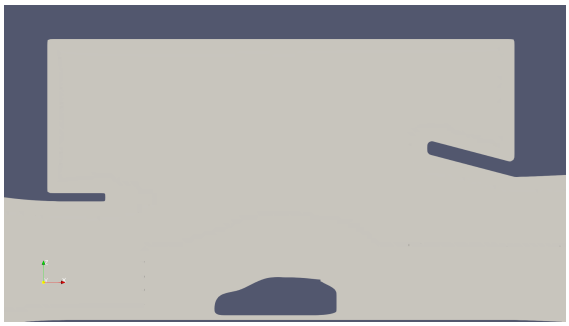


Fig. 3.20: configuration 4 - Tunnel domain

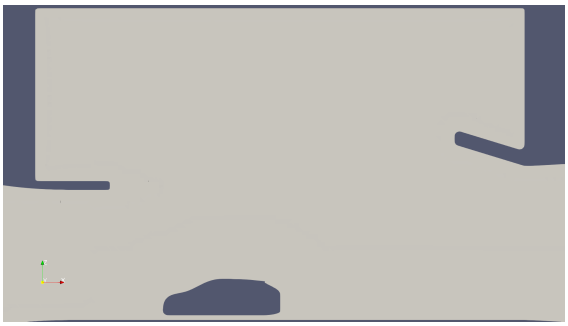


Fig. 3.21: configuration 5 - Tunnel domain

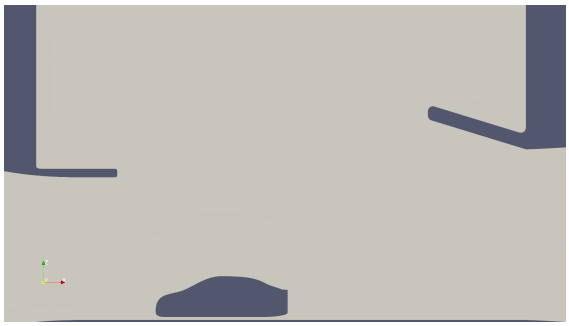


Fig. 3.22: configuration 6 - Tunnel domain

Fig. 3.23: configuration 7 - Tunnel domain

Fig. 3.24: configuration 8 - Tunnel domain

Fig. 3.25: configuration 9 - Tunnel domain

Fig. 3.26: configuration 10 - Tunnel domain

4. Results

4.1 Steady Simulations With Stationary Ground

In order to quantify the effect of the change in numerical schemes, reference cases have been ran from the previous end-point with the improved schemes presented in Section 3.3. By comparing the drag coefficient at the new convergence points with the one at the end of the previous simulations, the effect of numerical schemes can be isolated and studied.

In particular, 2 different types of vehicles have been chosen: a minivan model and a sedan model. Both of them have been simulated in 2 different domains, namely open-road and a generic tunnel geometry. This set of cases provided a good range of geometries on which the effects of the updated numerics have been quantified.

Figure 4.1 shows the percentage difference of drag coefficient between the current convergence point and the one at the end of previous simulations. It can be seen that differences depends not only on the vehicle but also on the computational domain of the simulation.

In the open-road domain, both configurations show a drag increase compared to the previous results, with the sedan model having a more significant change.

The minivan model shows a drag reduction for the case in the generic tunnel while the sedan model in the same domain shows a drag increase.

An additional case for the minivan vehicle model has been simulated using the AAWT wind tunnel geometry. This case allows to generally quantify the effect of numerical schemes when changing only the wind tunnel domain, while maintaining the same vehicle type.

By comparing results for the minivan between the generic tunnel and the AAWT tunnel we can see how the change in tunnel domain leads to changes in drag, suggesting that the effect of numerical schemes is significant when compared to previous results but its absolute value cannot be predicted, as it depends on multiple factors.

A trend for the effect of numerical schemes cannot be highlighted, suggesting that the study done before has to be carried out again, by implementing these changes for all future simu-

lations.

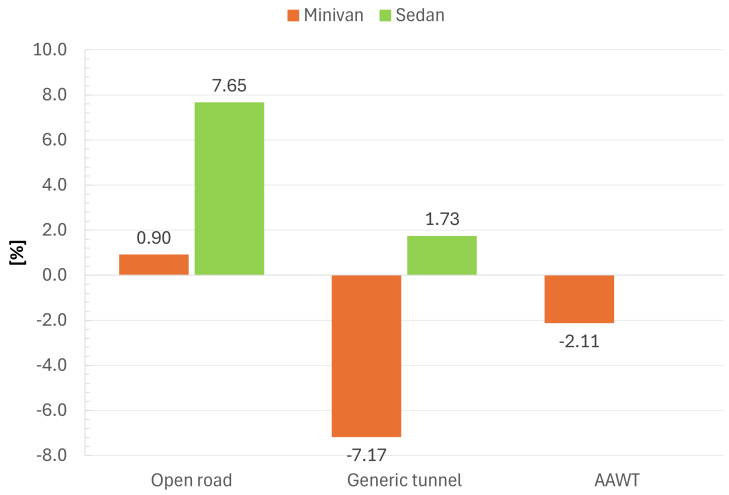


Fig. 4.1: Percentage difference between current and previous drag coefficients

Different vehicle shapes and different domains result in different pressure gradients which significantly affect the drag coefficient. Figure 4.2 shows the percentage difference between the current and previous drag considering the two contributions: pressure drag and viscous drag.

The changes in numerical schemes can be related to the behaviour of these two components. For the minivan case, the change in numerical schemes follows the same trend of the viscous drag: the higher the change in viscous contribution between current and previous cases and the greater is the effect of the numerics in absolute terms. The sedan model instead follows a trend more similar to the one highlighted by the pressure drag contribution: a greater absolute percentage difference for the pressure drag translates in a greater effect of the change in numerical schemes.

Even by considering this additional trends, the effect of the improved numerics cannot be predicted. Moreover, only general trends can be highlighted but the magnitude of these effects is strictly dependent on the vehicle and on the computational domain; no correlation between drag contributions and the sign of the effects of the numerics can be found comparing these quantities. This behaviour suggests once again that the change in numerics has to be implemented not only for robustness and convergence speed but also because its effect cannot

be predicted and affects each case in a different way.

(a)

(b)

Fig. 4.2: Percentage difference between current and previous drag coefficients. a) pressure drag contribution, b) viscous drag contribution

4.2 Steady Simulations With Moving Ground

A preliminary case including the moving ground system described in Section 3.5 has been carried out on the minivan model in the AAWT tunnel model, in order to quantify the effect of the inclusion of the moving ground system.
This case has been simulated with the improved numerics and the value at convergence has been compared to the value of the same simulation without the moving ground.

Table 4.1: Effect of moving ground system - Drag values

Configuration	**AAWT - Stationary ground**	**AAWT - Moving ground**
Minivan	0.2378	0.2389

Table 4.1 shows the results of the same wind tunnel domain both with and without the inclusion of the moving ground.
Typical uncertainty of drag coefficient is approximately ± 0.002 as reported by Walter et al. [34]. In this case, $\Delta C_d = C_{d,MG} - C_{d,SG} = 0.0011$; this value lies within the measurement uncertainty, suggesting that the effect of the implementation of the moving ground system can be difficult to validate experimentally.

While not translating in a huge difference in drag coefficient, the implementation of this system significantly affects the flow behaviour around the vehicle. Figure 4.3 shows the magnitude of normalized velocity on the symmetry plane for stationary and moving ground, respectively. Some differences in the wake region can be seen, with the moving ground case having a low-velocity region which occupies a greater portion of space, justifying the increase drag experienced by this configuration. In the case of stationary ground the wake region rejoins the flow in contact with the ground after about half a vehicle length downstream the model. This behaviour does not happen in case of moving ground, where the flow remains detached from the ground probably due to more momentum given by the flow coming from the underbody.

An additional slice at $z/l = 0.08$ is shown in Figure 4.4. It can be noticeable that for

the case with the moving ground, there is a region of flow deceleration on the whole side of the vehicle. This region is due to the fact that this second case implements wheel rotation which affects the behavior of the flow not only in correspondence of the wheels but also downstream. This region of reduced speed increases the aerodynamic resistance experienced by the model, again explaining the greater drag coefficient highlighted previously.

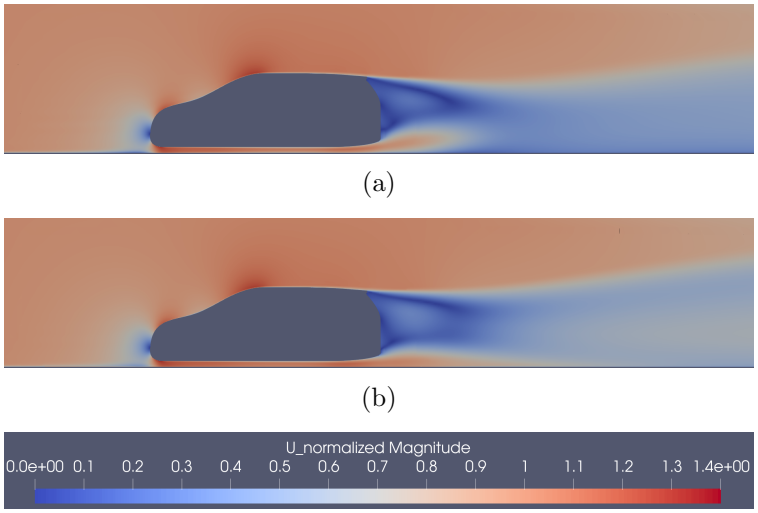


Fig. 4.3: U_{mean}/U_{∞} at $y/l = 0$. a) stationary ground, b) moving ground

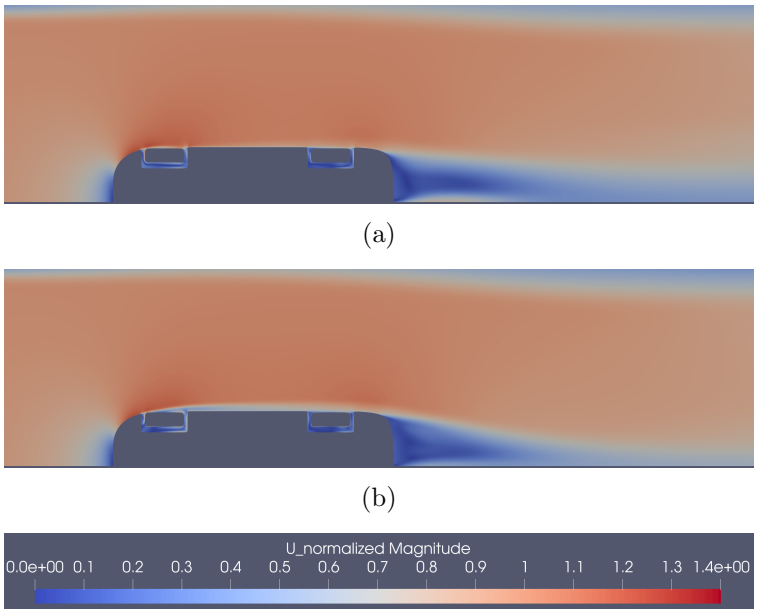


Fig. 4.4: U_{mean}/U_{∞} at $z/l = 0.08$. a) stationary ground, b) moving ground

Additional comparison is done considering the pressure coefficient distribution on the symmetry plane, as shown in Figure 4.5.

Minor differences can be noticed: the presence of the moving ground system reduces the pressure coefficient in correspondence of the diffuser (i.e. the end part of the underbody). While this behaviour would mostly affect the lift, the shape of the diffuser makes this pressure distribution affect also the drag: a negative c_p translates in a pressure vector directed outward from the surface, so exerting a force component directed opposite to vehicle motion.

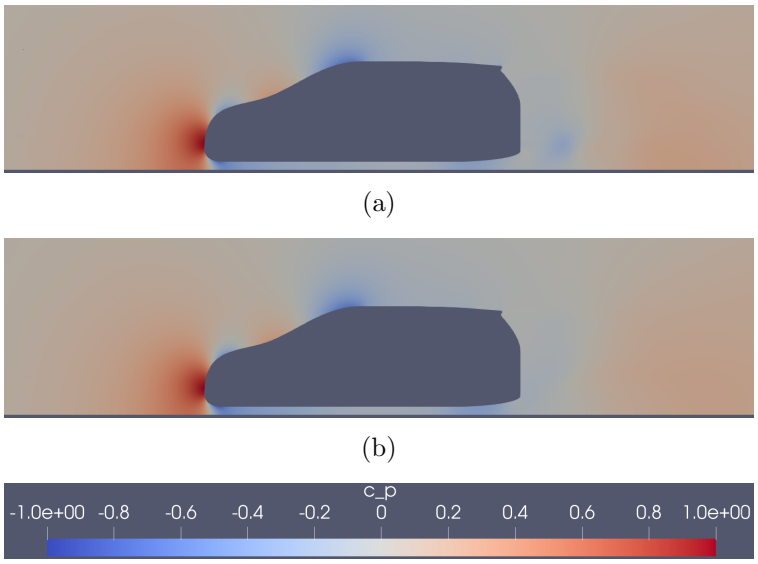


Fig. 4.5: c_p at $y/l = 0$. a) stationary ground, b) moving ground

The effect of wheel rotation can be seen more clearly looking at Figure 4.6 which shows the behaviour of turbulent kinetic energy, normalized using its inlet value.
It can be noticeable how for the moving ground case, there is an injection of energy coming from the front wheels, as expected from the previous discussion. This effect is progressing also downstream of the vehicle and affects the shape of the wake which appears to be more elongated in the longitudinal direction. Moreover, the stationary ground case highlights some high energy regions within the wake, probably due to some mixing vortices arising in that area.

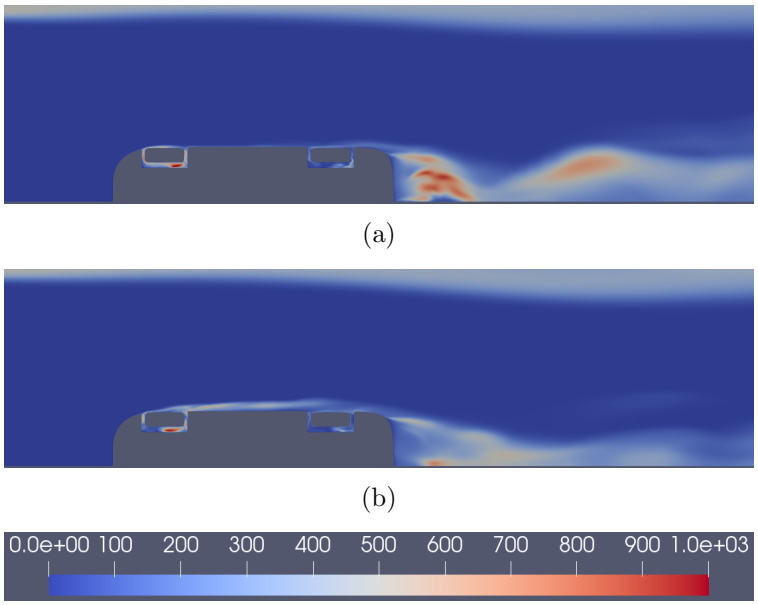


Fig. 4.6: k/k_{inlet} at $z/l = 0$. a) stationary ground, b) moving ground

To better assess the effects of the inclusion of the moving ground, the difference in drag contributions between stationary and moving ground has been computed and shown in Figure 4.7. Pressure drag decreases in the case of moving ground, due to the reduced dimensions of the wake region in the lateral direction and due to the flow momentum from the underbody of the vehicle which affects the behaviour of the flow on the back of the car. On the other hand, viscous drag increases as there is an additional region of flow deceleration on the side of the vehicle, increasing the wall shear stresses in that region. In this specific case, the increase in viscous drag contribution is the one affecting the total drag: the increase in this contribution outplays the decrease in pressure drag and consequently translates in an higher total drag.

In conclusion, the inclusion of moving ground leads to a small difference in terms of drag coefficient for the minivan model. As highlighted in Chapter 2, the literature does not show a specific trend in the simulations including this system but it is useful to implement it to reduce the differences between computational domain and real wind tunnels. Its implementation visibly affects the flow around the test vehicle, in terms of velocity, pressure and turbulence.

The integration of the system doesn't require any additional CAD geometries but is directly implemented through OpenFOAM functions, thus requiring minimum efforts. In addition, there is no increase in computational resources required so that the system can be easily implemented in all cases considered in the future.

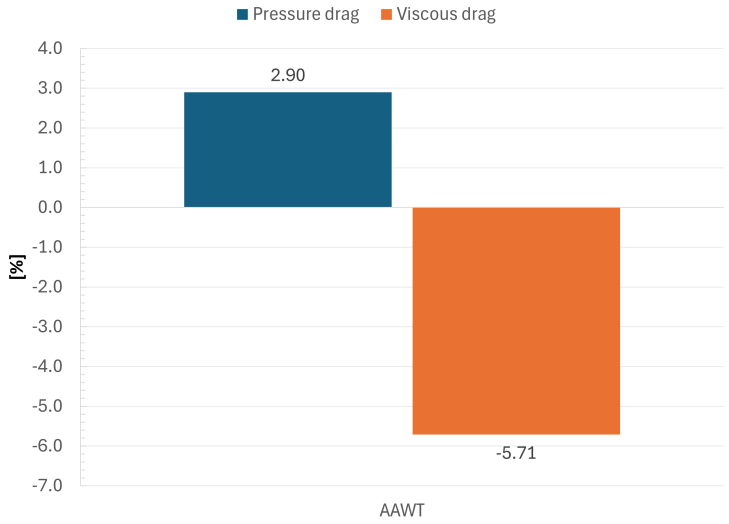


Fig. 4.7: Percentage difference in drag contributions between stationary and moving ground

4.3 Simulations for Parametric Study

Following the same approach pursued by Fujs [4], a parametric study has been carried out using the geometries generated as explained in Section 3.6.

All vehicle models have been simulated in 2 different computational environments: open-road and wind tunnel. An additional simulation of an empty wind tunnel has been carried out for each configuration; this latter being useful to retrieve dynamic pressure corrections and horizontal buoyancy corrections.

The corrected drag coefficient computed following the *classic* approach is

$$C_{d,q} = \frac{C_d + \Delta C_{d,\mathrm{HB}}}{q/q_\infty} \tag{4.1}$$

where C_d is the drag coefficient computed in the tunnel while $\Delta C_{d,\mathrm{HB}}$ is defined as:

$$\Delta C_{d,\mathrm{HB}} = \left(\frac{1.75}{A}\right)\left(\frac{V}{2}\right) G \tag{4.2}$$

where A is the frontal area of test vehicle, V is the volume of the test vehicle and G is the Glauert factor:

$$G = \left(\frac{dc_p}{dx}\right)_N + \left(\frac{dc_p}{dx}\right)_C \tag{4.3}$$

This factor is calculated by central difference with respect to the vehicle center by means of probes in the empty wind tunnel domain, at positions equivalent to vehicle front, center and rear. Figure 4.8 shows the location of probes for a generic configuration where the vehicle model has been included only for clarity.

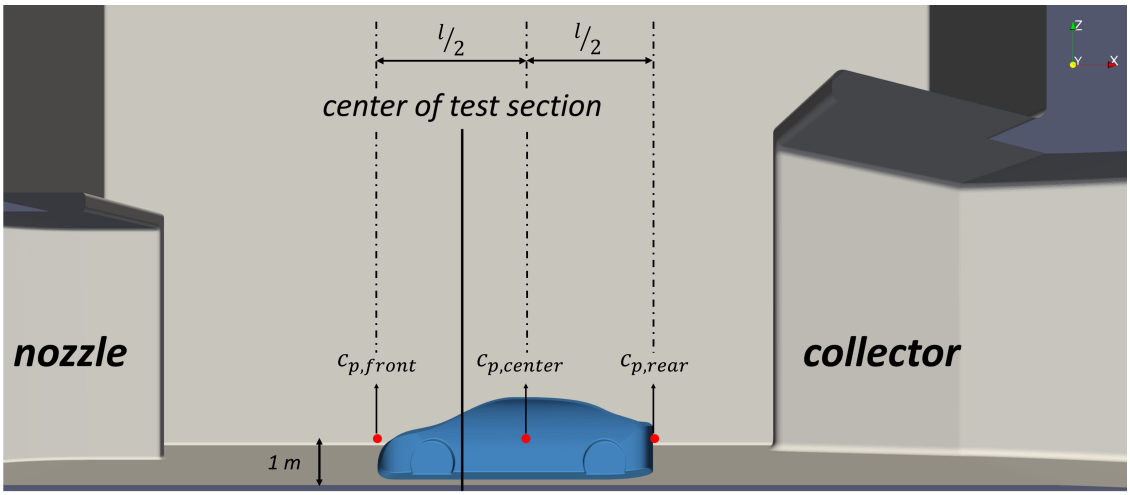


Fig. 4.8: Location of pressure probes in empty test chamber for a generic configuration

Referring back to Equation 4.1, $\frac{q}{q_\infty}$ is the dynamic pressure correction defined as:

$$\frac{q}{q_\infty} = (1 - \varepsilon_{\mathrm{QN}} + \varepsilon_{\mathrm{QP}} + \varepsilon_{\mathrm{S}} + \varepsilon_{\mathrm{N}} + \varepsilon_{\mathrm{C}})^2 \tag{4.4}$$

where each term is defined in Section C.2.

Results for drag coefficients for each configuration in the different domains are presented in Figure 4.9. Numerical values are included in Appendix C.3.

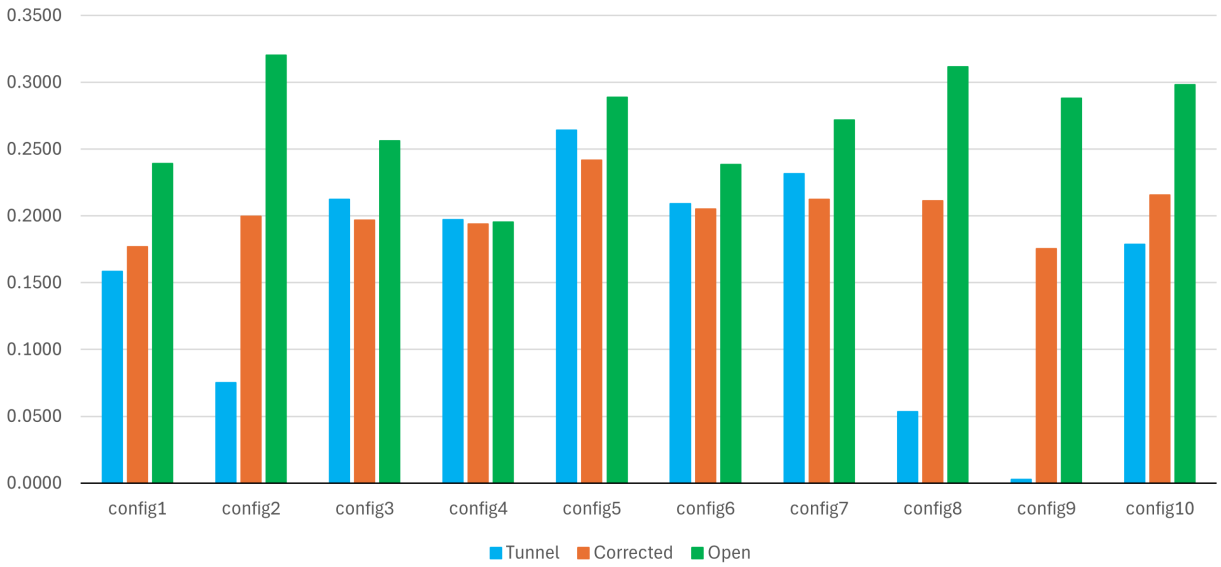


Fig. 4.9: Results for drag coefficient in the different domains

It is noticeable how some configurations, in particular *config2*, *config8* and *config9*, show a very low in-tunnel C_d when compared to the results of the other configurations.
Looking at Figure 3.18, Figure 3.24 and Figure 3.25 it can be seen that these configurations share a common detail: their position in the test chamber is really close to the collector. In this zone there is a favorable pressure gradient with an high pressure region upstream of the collector entry; this gradient reduces significantly the drag force experienced by the vehicle, translating in a lower drag coefficient computed for these specific cases.
In particular, Figure 4.10 show the pressure coefficient distribution at the back of the vehicle:

$$c_p = \frac{p - p_\infty}{1/2\,\rho\,U_\infty^2} \tag{4.5}$$

where p_∞ and U_∞ are taken in a point upstream of the vehicle where flow is undisturbed. ρ is a constant as cases are incompressible.

Configuration 1 is a more "standard" configuration where the vehicle is placed close to the center of the test section: for this configuration, c_p highlights values close to zero on the whole back of the vehicle, meaning that the pressure across that area is close to the free-stream value. On the other hand configuration 9 shows positive c_p values translating in a pressure force acting in the same direction of motion, reducing the drag experienced by

the vehicle.

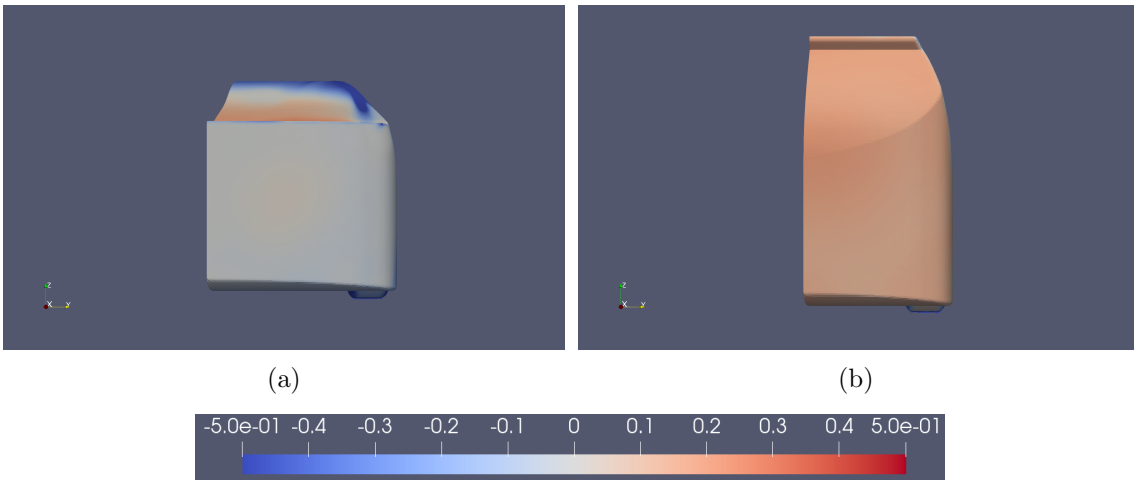


Fig. 4.10: c_p distribution on the back of the vehicle. a) configuration 1, b) configuration 9

While these 3 configurations are not realistic as in real tests the vehicle will never be placed so further in the test section, they appear to be useful to assess the performance of the *classic* pressure correction approach. Configuration 9 is an extreme case as the pressure gradient is so significant that it almost cancels out the drag force experienced by the vehicle. Even in this extreme case the correction approach is able to well correct the drag, bringing its value more in line with the the ones of other configurations.

It can be seen how the pressure corrections are able to significantly correct the configurations having a too low coefficient prior to this operation.
These corrections are very useful and perform well for all types of configurations, specifically the ones having a low drag coefficient in the tunnel domain prior to any correction; on the other hand there is no way to know if the corrected drag coefficient is the same as the one of the vehicle on open-road domain despite the good performance of the classic correction method. Indeed, it can be noticed how the drag coefficient computed for the open domain is constantly overshooting the corrected drag coefficient in the majority of the cases, suggesting that the pressure corrections are somehow limited in their performance. To justify the greater value in the open domain it is useful to look at the pressure coefficient distribution around the vehicle for both simulated domains, shown in Figure 4.11, where the lower bound of c_p scale has been limited to better show the differences.

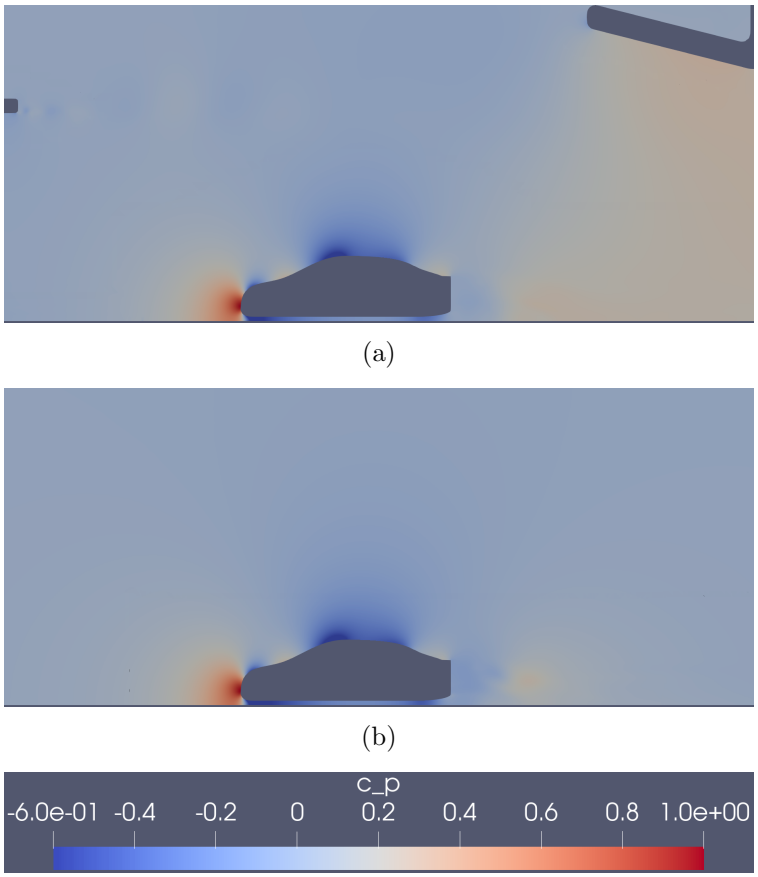


Fig. 4.11: c_p at $y/l = 0$ for configuration 1. a) tunnel domain, b) open domain

The first difference to be noticed is a slightly larger high pressure region at the front of the vehicle for the open domain: this translates to a higher pressure force working against the direction of vehicle motion, causing a higher drag coefficient for the vehicle.

A larger region of negative c_p can be seen at the trailing edge of the roof for the open domain: this distribution leads to a pressure force directed away from the body. In this area the pressure vector can be decomposed in the vehicle reference's frame: its horizontal component is opposite to the direction of motion, thus generating a force that increases the drag experienced by the vehicle. Some other differences in c_p distribution can be noticed underneath the car, where in the open domain a larger negative value region can be highlighted at the start and end sides of the underbody.

4.4 Improved Correction Method

To derive the improved correction the process presented in the previous work has been followed once again.

Fujs [4] found that drag coefficient computed for the simplified model consistently overshoots the one of the detailed model, allowing to retrieve a linear fit able to correlate the two values:

$$C_{d,\text{detailed}} = \frac{C_{d,\text{parametric}}}{mC_{d,\text{parametric}} + b + 1} \tag{4.6}$$

where $m = 0.22002$ and $b = 0.02715$.

This function is able to predict values for the detailed model with negligible error. By using this equation, a conversion factor describing the difference of drag between the parametric and detailed models can be retrieved:

$$f = C_{d,\text{parametric}} - C_{d,\text{detailed}} = C_{d,\text{parametric}} \left(1 - \frac{1}{mC_{d,\text{parametric}} + b + 1}\right) \tag{4.7}$$

The correction method previously used is based on the difference of drag coefficient between open domain and tunnel domain, corrected following the *classic* approach. This metric is computed from the results of the parametric study presented in Section 3.6, following Equation 4.9. Subtracting the factor f to C_d of the parametric model allows to have results for the detailed vehicle, improving the performance of the correction when considering real vehicles.

$$\Delta C_d = C^{open}_{d,\text{detailed}} - C^{tunnel}_{d,\text{detailed}} \tag{4.8}$$

$$= \left(C^{open}_{d,\text{parametric}} - f^{open}\right) - \left(C^{tunnel}_{d,\text{parametric}} - f^{tunnel}\right) \tag{4.9}$$

A set of 10 ΔC_d values is computed from the parametric study and used to retrieve a Radial Basis Function able to interpolate the scattered data. The idea behind this step is that those 10 points can be related to the values of PCs of each simulated geometry, leading to them being placed within the 4D PC space. RBF is used to retrieve a surface which interpolates

this points and that is able to return the value of ΔC_d for any configuration using as an input only the values of PCs, computed from the geometric parameters using the equations listed in Appendix B.6.

$$\Delta C_d = \mathrm{RBF}_n \left(PC_1, PC_2, PC_3, PC_4 \right) \tag{4.10}$$

Figure 4.12 shows the values of ΔC_d for the configurations of the parametric study, with exact values listed in Appendix C.4. It can be seen how all values are positive, highlighting once again that the open-road drag is always overshooting the corrected drag coefficient computed from the tunnel domain. This behaviour remains the same even after subtracting the f factor but the values are much closer to each other and differences remains in the range of about 50 to 75 drag counts for the majority of the cases (apart from the configurations pointed out before for their peculiarity of being close to the collector).

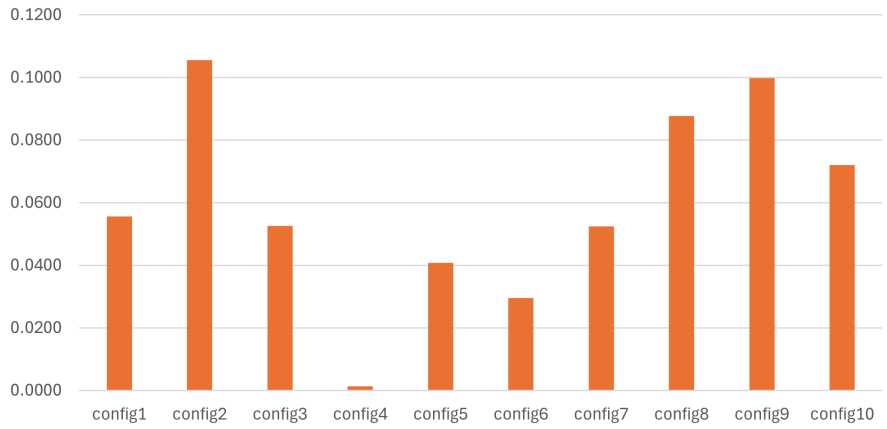


Fig. 4.12: ΔC_d for configurations of parametric study

Once the RBF function is obtained, it can be applied on the experimental dataset used initially in order to retrieve the values of ΔC_d for those configurations and to check the performance of the correction in terms of mean value and standard deviation.
These values are added to the drag coefficient computed experimentally and corrected following the classic correction approach, leading to the results shown in Table 4.2. Values of ΔC_d for the experimental dataset are available in Appendix C.5.

At this point results can be compared to the other correction methods, applied on the

Table 4.2: C_d for experimental dataset corrected with new RBF

Configuration	**Tunnel 1**	**Tunnel 2**	$\mathbf{C_{d,q,T1} - C_{d,q,T2}}$
vehicle 1	0.3838	0.3323	0.0516
vehicle 2	0.3953	0.3367	0.0586
vehicle 3	0.3818	0.3364	0.0453
vehicle 4	0.4155	0.3459	0.0697
vehicle 5	0.4487	0.3659	0.0827
		μ	0.0616
		σ	0.0149

same experimental dataset. RBF_{10} is the label used for the new correction method. Results are shown in Figure 4.13 while numerical values for each correction method are given in Appendix C.6.

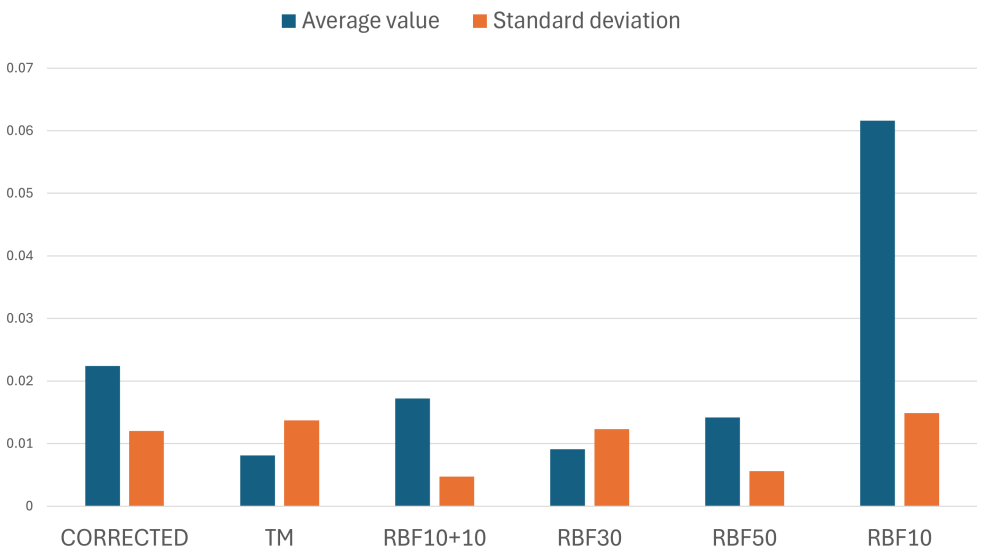


Fig. 4.13: $C_{d,q,T1} - C_{d,q,T2}$ for different correction methods

Considering each correction individually, RBF_{10} does not provide any improvement compared to the other approaches: indeed its standard deviation is the highest among all considered methods.

RBF_{10} and RBF_{30} share a similar sampling approach, where the points are inter-tunnel sam-

plings (i.e. not having any samples in the tunnel geometries considered in the experimental dataset). By comparing these two correction approaches we can see how the standard deviation of RBF_{10} is close to the one computed for RBF_{30} despite using only a third of the sampling points. While the mean value is much greater in the case of RBF_{10}, this is not of major interest as its effect can be easily canceled out by offsetting all results by μ.

Standard deviation is much more important as it quantifies the spread of the corrected drag difference across the considered configurations. In an ideal case where $\sigma = 0$, the drag difference between the two tunnels is constant across all vehicles, allowing for a simple offset correction that will lead to the same drag coefficient for a given vehicle independently from the tunnel geometry in which the test is performed.

This observation suggests that increasing the sampling density in the central region of the PC space does not provide significant benefits for the correction approach, since the vehicles to which the correction is applied lie at the extreme ends of the PC space (i.e. $PC_1 = \pm 3.2863$), as it can be seen from Figure 4.14.

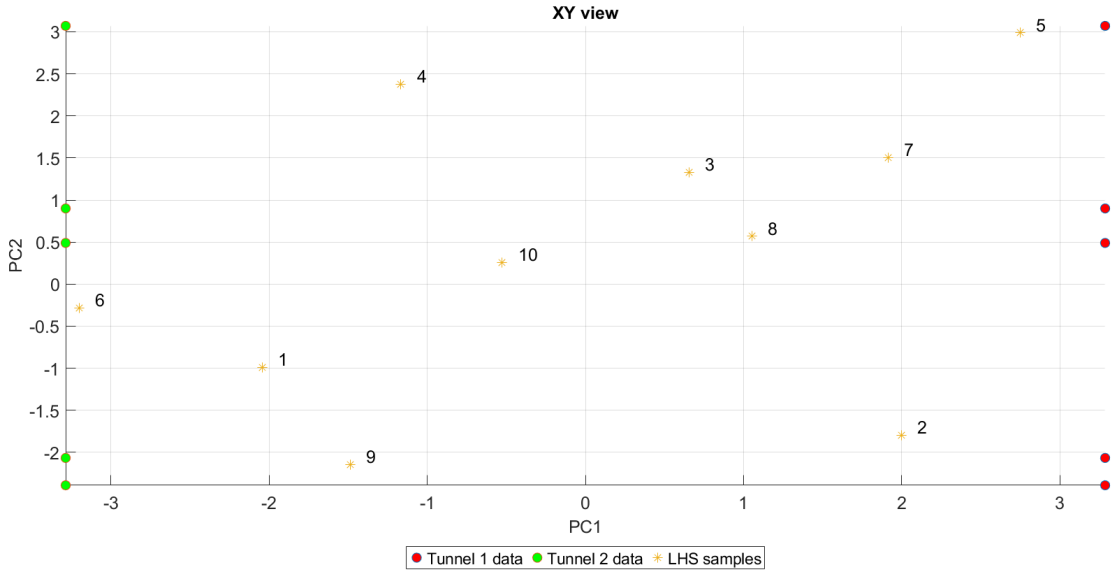


Fig. 4.14: Position of sampling points in the PC space

Previously defined RBFs can be used in order to improve the performance of RBF_{10}. In particular, RBF_{30} is based on 30 points distributed within the PC space, away from the edges of the domain and RBF_{10+10} is based on 20 points (10 for each tunnel geometry of the experimental dataset) placed exactly at the two extremes of PC_1. RBF_{50} is a function

based on the combination of these two.

Since the points of the experimental dataset lie on the edges of the PC space, a finer sampling in correspondence of these regions is beneficial for the performance of the correction when applied to the same dataset used previously. Figure 4.15 shows a comparison between the correction methods, including a new RNF defined from the combination of some of them.

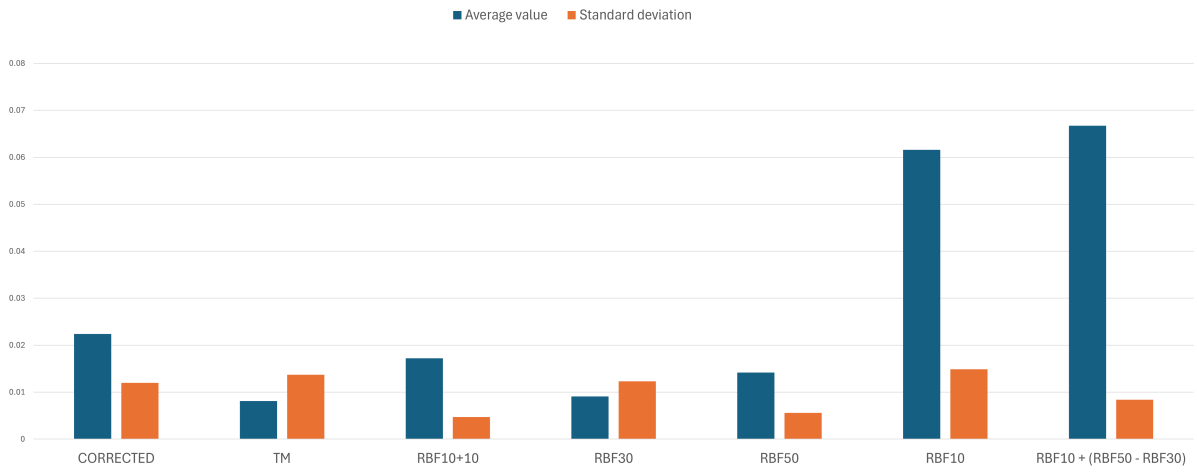


Fig. 4.15: $C_{d,q,T1} - C_{d,q,T2}$ including the combination of different correction methods

$(\mathrm{RBF}_{50} - \mathrm{RBF}_{30})$ is calculated to isolate the effect of the additional 20 sampling points at the edges of the PC domain, while preserving the behavior of the function in the central region.

Adding this residual correction to RBF_{10} leads to a significant improvement in the results: standard deviation is reduced by **43%** when compared to the one of RBF_{10} only. This improvement is likely because $(\mathrm{RBF}_{50} - \mathrm{RBF}_{30})$ is close to zero far from the edges of PC space, yet maintains a smooth blend across the entire domain. The results of this combined correction confirms that the sampling strategy used for RBF_{10} is well designed and can be more beneficial if additional experimental data is available.

A similar combination using RBF_{10+10} results in worse performance than using RBF_{10} alone. This is due to the fact that RBF_{10+10} contains no information about ΔC_d in the central region of the PC space and therefore fails to blend properly with RBF_{10}.

5. Conclusion

5.1 Summary

The work presented in this thesis aimed at improving the simulation models and drag predictions for automotive bluff bodies in open-jet wind tunnels.
CFD RANS steady-state simulations have been the focus, with numerous changes that have been implemented in order to achieve better accuracy of the results while increasing robustness of the cases and convergence speed required by each simulation. Significant work has been done on top of what was presented previously in the literature; first of all, being the work based on an automatic approach for the meshing of the computational grid, grid quality was often not excellent and numerical schemes have thus been changed to better perform in case of high skewness and high non-orthogonality. It was noticed how these changes do not present any clear trend, with the change in drag coefficient depending on both the vehicle and on the simulation domain. In all simulated cases the effect was not negligible, suggesting that the approach used can have significant effects on the results.

Subsequently, an improvement of the fidelity of the models used for simulations has been implemented. This improvement refers to the implementation of a moving ground system in the tunnel domain used for simulations. This system is defined starting from the same system available in the AAWT tunnel of Stellantis US and the approach has been generalized in order to implement the same system in the same way independently from the size of the car or the geometry of the tunnel, thus ensuring reproducibility.
The presented approach also includes the implementation of rotating wheels for the vehicle model. The simplicity of the model allowed for the rotation of these being implemented by using specific boundary conditions, without requiring the usage of MRFs. Also for this step a generalization is given in order to enable the inclusion of this feature by only using vehicle dimensions.

Finally, a parametric study has been performed using the improved models. A reduced number of configurations has been selected due to time limitations but results were promising, suggesting that the proposed approach can be used for future applications.

5.2 Thesis Contributions

The work pursued in this thesis contributes in many ways to the automotive literature. The main contribution is the update to a complete automatic approach, able to generate simulation geometries by only starting from data widely available for each vehicle and/or open-jet tunnel configurations. This tool allows to not only save time when performing CFD simulations but also to have a good estimation of the real flow behaviour around an automotive bluff body, while maintaining the advantage of RANS simulations without sacrificing accuracy. Specifically, robustness enhancements have been integrated so that the models successfully generate over a large range of inputs. Vehicle and tunnel models have been modified to allow the implementation of a rolling road system in the computational domain. The generalization of each step allows the use of the same model independently from the geometry of interest, increasing the capability of the approach to satisfy a wide range of cases.

In addition, the approach followed for the correction highlighted the importance of the sampling strategy. This aspect has to be designed in function of the aim of the correction. Generally speaking, better performance for a given dataset is found when sampling points are already close to the geometry used for the experiments; a more universal approach is more difficult to achieve but performance can be highly satisfactory when compared to the current state of art, thus offering powerful tools to reduce the tunnel-to-tunnel drag variation.

5.3 Future Work

The study presented in this thesis highlights some trade-offs due to the limited available time for the work.

The effect of numerical schemes can be compared to experimental data in order to check if their change improves the correlation between CFD and real tests. Moreover, further validation on experimental data should be carried out in order to validate the improvements done on the simulations.

The correction approach can be improved by increasing the number of reference geometries in the initial dataset. This increase in initial information will lead to a more universal improved correction rather than one tailored for only the tunnel geometries presented.
An increase in the number of samples, focused on a finer sampling in the PC regions closer to the PC values of the reference dataset, will help in defining a more robust correction when using the same reference dataset used in this thesis.

Finally, the performance of the approach can be better assessed by using higher fidelity simulations, for example by performing unsteady simulations.

REFERENCES

[1] Alamaan Altaf, Ashraf A. Omar, and Waqar Asrar. Passive drag reduction of square back road vehicles. *Journal of Wind Engineering and Industrial Aerodynamics*, 134: 30–43, 2014. ISSN 0167-6105.

[2] Michal Fabian, Róbert Huňady, František Kupec, and Tomáš Mlaka. Effect of the aerodynamic elements of the hatchback tailgate on the aerodynamic drag of the vehicle. *Advances in Science and Technology Research Journal*, 16:73–87, 12 2022.

[3] Elton Nyoni and Primrose Chigumba. Aerodynamics and its role in enhancing fuel efficiency in automotive engineering. *International Journal of Automobile Engineering*, 5(2):32–35, 2024. ISSN 2707-8213.

[4] Matthew Fujs. Parametric automobile and open-jet wind tunnel models and their application to improved drag coefficient corrections. *Electronic Theses and Dissertations*, 9330, 2023. URL https://scholar.uwindsor.ca/etd/9330.

[5] Emil Ljungskog, Simone Sebben, and Alexander Broniewicz. Inclusion of the physical wind tunnel in vehicle cfd simulations for improved prediction quality. *Journal of Wind Engineering and Industrial Aerodynamics*, 197, 2020. ISSN 0167-6105.

[6] Alessandro Bottaro. Meccanica dei fluidi, 2019. University course taught at Univeristà di Genova.

[7] James Julian, Tulus Hidayat Yusanto, Adi Winarta, Fitri Wahyuni, Muhammad Ilham Adhynugraha, and Fadilah Hasim. Numerical analysis of 6-dof independent external balance for subsonic wind tunnel. *Engineering Science and Technology, an International Journal*, 54:101704, 2024.

[8] Jim Ross, Matthew Rhode, Bryan Falman, Karl Edquist, Mark Schoenenberger, Greg Brauckmann, William Kleb, Thomas West, Stephen Alter, and David Witte. Evaluation

of cfd as a surrogate for wind-tunnel testing for mach 2.4 to 4.6 - project overview. 08 2021.

[9] S. R. Ahmed, G. Ramm, and G. Faltin. Some salient features of the time -averaged ground vehicle wake. *SAE Transaction*, 93, 1984.

[10] Angelina I. Heft, Thomas Indinger, and Nikolaus A. Adams. Introduction of a new realistic generic car model for aerodynamic investigations. *SAE Technical Paper*, 2012.

[11] Mohamed Sukri Mat Ali, Jafirdaus Jalasabri, Anwar Sood, Shuhaimi Mansor, Haziqah Shaharuddin, and Sallehuddin Muhamad. Wind noise from a-pillar and side view mirror of a realistic generic car model, driaver. *International Journal of Vehicle Noise and Vibration*, 14:38, 01 2018.

[12] Neil Ashton and Alistair Revell. Comparison of RANS and DES methods for the drivaer automotive body. *SAE Technical Papers*, 2015, 04 2015.

[13] Thomas Schuetz. *Aerodynamics of Road Vehicles (5th Edition)*. SAE International, 2016.

[14] Edzard Mercker, Gerhard Wickern, and Jochen Weidemann. Contemplation of nozzle blockage in open jet wind-tunnels in view of different 'q' determination techniques. *SAE Transactions*, 106:283–292, 1997.

[15] E. Mercker and K.R. Cooper. A two-measurement correction for the effects of a pressure gradient on automotive, open-jet, wind tunnel measurements. *SAE Transactions*, 115, 2006.

[16] T. Lounsberry and J. Walter. Practical implementation of the two-measurement correction method in automotive wind tunnels. *SAE International Journal of Passenger Cars–Mechanical Systems*, 8(2):676–686, 2015.

[17] Oliver Fisher. Investigation of correction methods for interference effects in open-jet wind tunnels. *Springer Fachmedien Wiesbaden*, 2018.

[18] Adithya Gopal. Stellantis invests in wind tunnel technology for evs. *Automotive Testing Technology International*, October 2024. URL https://www.automotivetestingtechnologyinternational.com/news/aerodynamics/stellantis-invests-in-wind-tunnel-technology-for-evs.html.

[19] Rasmus Bro and Age K. Smilde. Principal component analysis. *Analytical Methods*, 6: 2812–2831, 2014.

[20] Wei-Liem Loh. On Latin hypercube sampling. *The Annals of Statistics*, 24(5):2058 – 2080, 1996.

[21] Martin D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[22] Siniša Krajnović and Lars Davidson. Influence of floor motions in wind tunnels on the aerodynamics of road vehicles. *Journal of Wind Engineering and Industrial Aerodynamics.*, 2005. ISSN 0167-6105.

[23] Shibo Wang, Terence Avadiar, Mark C. Thompson, and David Burton. Effect of moving ground on the aerodynamics of a generic automotive model: The drivaer-estate. *Journal of Wind Engineering and Industrial Aerodynamics*, 195, 2019. ISSN 0167-6105.

[24] Vesselina Roussinova. Advanced fluid mechanics MECH 8290-23, A.Y. 2024-2025. University course taught at University of Windsor.

[25] Chunhui Zhang, Charles Patrick Bounds, Lee Foster, and Mesbah Uddin. Turbulence modeling effects on the cfd predictions of flow over a detailed full-scale sedan vehicle. *Fluids*, 4(3), 2019.

[26] Quentin Carré. Turbulence modelling for CFD. *Eindhoven University of Technology*, 2023.

[27] SimScale. Mesh quality, 2024. URL https://www.simscale.com/docs/simulation-setup/meshing/mesh-quality/.

[28] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.6 Overview of discretisation*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/overview-of-discretisation.

[29] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.10 Upwind scheme*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/upwind-scheme.

[30] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.14 Linear upwind scheme*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/linear-upwind-scheme.

[31] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.22 Bounded advection discretisation*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/bounded-advection-discretisation.

[32] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.11 Limited advection schemes*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/limited-advection-scheme.

[33] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.23 Recommended discretisation schemes*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/recommended-discretisation-schemes.

[34] J. Walter, V. Canacci, R. Rout, W. Koester, and M. Simon. Uncertainty analysis of aerodynamic coefficients in an automotive wind tunnel. SAE Technical Paper 2005-01-0870, SAE International, 2005.

[35] MathWorks. *zscore*. The MathWorks, Inc., . URL https://www.mathworks.com/help/stats/zscore.html.

[36] MathWorks. *lhsdesign*. The MathWorks, Inc., . URL https://www.mathworks.com/help/stats/zscore.html.

[37] MathWorks. *lsqnonlin*. The MathWorks, Inc., . URL https://www.mathworks.com/help/stats/zscore.html.

[38] OpenCFD Ltd. *Forces function object*. OpenFOAM Foundation, 2021. OpenFOAM User Guide v2112.

[39] Nathan A. Rooy. parea: The easiest way to calculate the projected/frontal area of an stl. Python package (PyPI), 2020. Released December 17, 2020.

[40] Chris Greenshields and Henry Weller. *Notes on Computational Fluid Dynamics: General Principles – 3.12 Useful TVD schemes*. CFD Direct Ltd, 2022. URL https://doc.cfd.direct/notes/cfd-general-principles/useful-tvd-schemes.

APPENDICES

A. Appendix A

A.1 Breakdown of model parameters

Table A.1: Parameters for simplified model [4]

Description	**Symbol**
Tunnel	
Test section length	L
Test section width	W
Test section height	H
Nozzle exit width	W_N
Nozzle exit height	H_N
Nozzle depth into test section	D_N
Collector exit width	W_C
Collector exit height	H_C
Collector depth into test section	D_C
Collector angle	α
Diffuser angle	β
Vehicle placement	x
Vehicle	
Length	l
Width	w
Height	h
Internal volume, passenger compartment and trunk	v
Front overhang	o_f
Rear overhang	o_r
Front wheel well height	h_f
Front wheel well height	h_r
Ride height	r

B. Appendix B

B.1 Boundary conditions for wind tunnel domain

Table B.1: Boundary conditions for wind tunnel domain

Patch	**U**	**p**	**k**	ω	ν_t
Inlet	fixedValue	zeroGradient	fixedValue	fixedValue	calculated
Outlet	zeroGradient	fixedValue	zeroGradient	zeroGradient	calculated
Walls	noSlip	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Nozzle bottom surface and final tunnel portion	slip	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Vehicle model and wheels	noSlip	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Wheels	rotatingWallVelocity	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Moving ground system	movingWallVelocity	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction

B.2 Boundary conditions for open domain

Table B.2: Boundary conditions for open road domain

Patch	**U**	**p**	**k**	ω	ν_t
Inlet	fixedValue	zeroGradient	fixedValue	fixedValue	calculated
Outlet	zeroGradient	fixedValue	zeroGradient	zeroGradient	calculated
Ground	movingWallVelocity	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Vehicle model	noSlip	zeroGradient	kqRWallFunction	omegaWallFunction	nutUSpaldingWallFunction
Wheels	rotatingWallVelocity	zeroGradient	kqRWallFucntion	omegaWallFunction	nutUSpaldingWallFunction

B.3 Grid quality metrics for a generic configuration in the tunnel domain

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2312                                  |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : _1d8f0d55f7-20231221 OPENFOAM=2312 version=v2312
Arch   : "LSB;label=32;scalar=64"
Exec   : checkMesh
Date   : Jul 05 2025
Time   : 10:38:40
Host   : nl20101
PID    : 134413
I/O    : uncollated
Case   : /home/philpess/scratch/Tunnel/001_config1_TUNNEL
nProcs : 1
trapFpe: Floating point exception trapping enabled (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
    (fileModificationSkew 5, maxFileModificationPolls 20)
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Check mesh...

Time = 0

Mesh stats
    points:           16218770
    faces:            46932873
    internal faces:   46036753
    cells:            15362055
    faces per cell:   6.0519
    boundary patches: 11
    point zones:      1
    face zones:       0
```

```
    cell zones:       0

Overall number of cells of each type:
    hexahedra:     14853766
    prisms:        107194
    wedges:        10265
    pyramids:      1
    tet wedges:    8236
    tetrahedra:    46
    polyhedra:     382547
    Breakdown of polyhedra by number of faces:
        faces   number of cells
            4   3897
            5   4173
            6   49449
            7   90698
            8   11991
            9   177173
           10   176
           11   9
           12   39760
           13   1
           14   1
           15   5044
           18   174
           21   1

Checking topology...
    Boundary definition OK.
    Cell to face addressing OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).

Checking patch topology for multiply connected surfaces...
                  Patch    Faces   Points  Surface topology
               symmetry   125896   130039  ok (non-closed singly connected)
                  inlet    17378    18080  ok (non-closed singly connected)
                 outlet     8740     9085  ok (non-closed singly connected)
              wallsslip   109606   112470  ok (non-closed singly connected)
            wallsnoslip   439427   446389  ok (non-closed singly connected)
            vehicleBody   162034   163686  ok (non-closed singly connected)
            frontWheels    10496    11117  ok (non-closed singly connected)
             rearWheels    10383    11015  ok (non-closed singly connected)
```

```
            centerBelt    10633    11050  ok (non-closed singly connected)
       frontLeftBelt      745      850  ok (non-closed singly connected)
        rearLeftBelt      782      881  ok (non-closed singly connected)
                ".*"   896120   906883      ok (closed singly connected)


Checking faceZone topology for multiply connected surfaces...
    No faceZones found.

Checking basic cellZone addressing...
    No cellZones found.

Checking basic pointZone addressing...
           PointZone  PointsBoundingBox
        frozenPoints      24(-13.2522 4.11526 -0.613871) (-13.1801 4.1878 -0.529353)

Checking geometry...
    Overall domain bounding box (-32.6556 0 -4.17379) (66.6684 7.34801 11.069)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (1.12454e-15 -7.86838e-14 4.28925e-15) OK.
    Max cell openness = 3.72876e-15 OK.
    Max aspect ratio = 423.681 OK.
    Minimum face area = 1.08692e-07. Maximum face area = 0.478868.  Face area magnitudes OK.
    Min volume = 1.69898e-10. Max volume = 0.236236.  Total volume = 4945.23.  Cell volumes
        OK.
    Mesh non-orthogonality Max: 87.1008 average: 6.11241
   *Number of severely non-orthogonal (> 70 degrees) faces: 14357.
    Non-orthogonality check OK.
  <<Writing 14357 non-orthogonal faces to set nonOrthoFaces
    Face pyramids OK.
 ***Max skewness = 4.56255, 7 highly skew faces detected which may impair the quality of
     the results
  <<Writing 7 skew faces to set skewFaces
    Coupled point location match (average 0) OK.

Failed 1 mesh checks.

End
```

B.4 fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2312                                  |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
FoamFile
{
    version         2;
    format          ascii;
    class           dictionary;
    location        "system";
    object          fvSchemes;
}

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         cellLimited leastSquares 1;
}

divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwindV limited;
    div(phi,k)      Gauss Minmod;
    div(phi,omega)  Gauss Minmod;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
    div(div(phi,U)) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
```

```
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

wallDist
{
    method          meshWave;
}


// ************************************************************************* //
```

B.5 Python code for geometries generation

```
#!/usr/bin/env python

import os
import sys
import salome
import salome_notebook
from SketchAPI import *
from salome.shaper import model
import GEOM
from salome.geom import geomBuilder
import math
import SALOMEDS
import shutil

salome.salome_init()
notebook = salome_notebook.NoteBook()

#
    ----------------------------------------------------------------------------------------------------------------
# Change the following parameters to suit your needs. Note, Salome is a tricky program, and
    sometimes rerunning the script will produce different results.
# It may help to simply run the script again if unfathomable results appear.
#
#
# folder --> output folder path, may need to be created.
# stlRefinement --> minimum relative refinement level, lower being more refined
#
# TunnelParam --> the parameters for the wind tunnel, listed in order below
#    L - Test Section Length
#    W - Test Section Width
#    H - Test Section Height
#    W_n - Nozzle Width
#    H_n - Nozzle Height
#    D_n - Nozzle Depth into Test Section
#    W_c - Collector Width
#    H_c - Collector Height
#    D_c - Collector Depth into Test Section
#    Alpha - Collector Angle
#    Beta - Diffuser Angle
#    TunnelPlacement - Placement of the vehicle away from the center of the test section
    (positive is towards the nozzle)
#
# VehicleParam --> the parameters for the Vehicle, listed in order below
```

```
#    l - Length
#    w - Width
#    h - Height
#    v - Volume (trunk volume + passenger volume)
#    o_f- Front Overhang
#    o_r - Rear Overhang
#    h_f - Front Wheel Well Height
#    h_r - Rear Wheel Well Height
#    r - Ride Height
#
# The part generation sometimes runs into errors that can be seen in the object browser in
    Salome. Typically, finding where the error occurs and inspecting the sketch or element
    will lead
# to a possible solution. The steps below guide you to solutions for ones experienced
    previously.
#
# filletRadius --> small fillets on the outside of the vehicle tend to have problems. If
    errors noticed for Fillet_3 in Salome, try modifying the decimal value by +- 0.002.
#                  If you right-click and edit the fillet, you can change the decimal value
    within Salome until the sketch turns green, signalling it works. Then modify the value
#                  here in the code to match.
# bigFilletRadius --> large fillets at the rear of the vehicle can also be a cause for
    problems as the edges may intersect with where the spoiler sits.
#                     If Fillet_3 still has issues and cannot be fixed by modifying the
    number above, try modifying this decimal value by +- 0.01. When it is clear that the
    edges do not
#                     interfere with each other, you may have to return to the previous
    value and modify it again until it works.
# spline --> a spline used to create the spoiler sometimes does not work as intended. This
    is noticable in Salome when the sketches show errors, specifically vehicle Sketch_1.
#            This can be worked around by changing the value of this spline. Typically
    changing to either 2 or 2.5 works. However, sometimes you may have to try some in
    between values.
#            If all else fails, try values just below and above 2 and 3 respectively,
    changing by 0.1 at a time.

stlRefinement = 1e-5

#
    ----------------------------------------------------------------------------------------------------
# PARAMETRIC GEOMETRIES
#
    ----------------------------------------------------------------------------------------------------
configN = 10
```

```
if configN == 1:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config1"
    TunnelParam = [18.6913, 0.6964, 0.5922, 0.3951, 0.2509, 0.1179, 0.4764, 0.3039, 0.1969,
        14.2288, 2.6911, -0.2329]
    VehicleParam = [4.7276, 0.4219, 0.3108, 0.0275, 0.1809, 0.17, 0.1462, 0.1448, 0.0203]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.05*l"
    spline = 3

elif configN == 2:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config2"
    TunnelParam = [17.3768, 0.6554, 0.5227, 0.3242, 0.2138, 0.0664, 0.4269, 0.2653, 0.1371,
        10.9423, 1.3767, -3.355]
    VehicleParam = [4.306, 0.3219, 0.35, 0.035, 0.17, 0.17, 0.1404, 0.1402, 0.0329]
    filletRadius = "0.003*l"
    bigFilletRadius = "0.03*l"
    spline = 2.5

elif configN == 3:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config3"
    TunnelParam = [19.3941, 0.7375, 0.6207, 0.4211, 0.261, 0.1404, 0.4645, 0.3114, 0.1672,
        15.9852, 3.394, 1.4359]
    VehicleParam = [5.1156, 0.4278, 0.3365, 0.032, 0.1768, 0.1875, 0.16, 0.16, 0.0304]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.05*l"
    spline = 3

elif configN == 4:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config4"
    TunnelParam = [18.8835, 0.7081, 0.5998, 0.4019, 0.2534, 0.1239, 0.4722, 0.3056, 0.1869,
        14.7092, 2.8834, 0.2235]
    VehicleParam = [4.9349, 0.43, 0.35, 0.035, 0.2094, 0.17, 0.16, 0.16, 0.04]
    filletRadius = "0.003*l"
    bigFilletRadius = "0.03*l"
    spline = 2.7

elif configN == 5:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config5"
    TunnelParam = [19.7839, 0.7627, 0.6355, 0.4341, 0.2654, 0.1522, 0.453, 0.314, 0.1433,
        16.9595, 3.784, 2.3617]
```

```
    VehicleParam = [4.7388, 0.43, 0.35, 0.035, 0.17, 0.17, 0.16, 0.16, 0.04]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.05*l"
    spline = 2.9

elif configN == 6:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config6"
    TunnelParam = [19.799, 0.7142, 0.6581, 0.4651, 0.2907, 0.1656, 0.5, 0.3477, 0.2,
        16.9988, 3.7986, 2.3981]
    VehicleParam = [5.3385, 0.422, 0.3312, 0.0265, 0.2066, 0.1993, 0.1566, 0.1556, 0.0226]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.05*l"
    spline = 3

elif configN == 7:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config7"
    TunnelParam = [19.5542, 0.7494, 0.6261, 0.4255, 0.262, 0.1448, 0.4566, 0.3114, 0.1523,
        16.3853, 3.5542, 1.8161]
    VehicleParam = [4.9833, 0.43, 0.35, 0.0312, 0.1969, 0.17, 0.16, 0.16, 0.0259]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.05*l"
    spline = 2.6

elif configN == 8:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config8"
    TunnelParam = [17.1342, 0.65, 0.5132, 0.3156, 0.2107, 0.0588, 0.4323, 0.2632, 0.1493,
        10.336, 1.1341, -3.931]
    VehicleParam = [3.8731, 0.3567, 0.3464, 0.0325, 0.1845, 0.178, 0.1599, 0.1599, 0.0398]
    filletRadius = "0.003*l"
    bigFilletRadius = "0.04*l"
    spline = 2.2

elif configN == 9:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config9"
    TunnelParam = [17.2749, 0.65, 0.5244, 0.3285, 0.219, 0.0665, 0.455, 0.2732, 0.1831,
        10.688, 1.2747, -3.5968]
    VehicleParam = [4.5122, 0.3339, 0.347, 0.035, 0.17, 0.17, 0.14, 0.14, 0.02]
    filletRadius = "0.005*l"
    bigFilletRadius = "0.02*l"
    spline = 2.7
```

```
elif configN == 10:
    folder = "C:/Users/phili/Desktop/University of Windsor - Documents/Thesis -
        CFD/SalomeGeometries/Parametric study/config10"
    TunnelParam = [18.2782, 0.6844, 0.5699, 0.3722, 0.2388, 0.1014, 0.459, 0.2911, 0.1749,
        13.196, 2.2781, -1.2141]
    VehicleParam = [4.5036, 0.3618, 0.3364, 0.0346, 0.1798, 0.187, 0.1588, 0.1589, 0.0381]
    filletRadius = "0.003*l"
    bigFilletRadius = "0.02*l"
    spline = 2.9

else:
    print("You entered an invalid configuration number, try again!")

#
    ------------------------------------------------------------------------------------------------------------
# FOLDER CREATION
#
    ------------------------------------------------------------------------------------------------------------
if not os.path.exists(folder):
    os.mkdir(folder)
if not os.path.exists(folder + "/STL"):
    os.mkdir(folder + "/STL")
if not os.path.exists(folder + "/XAO"):
    os.mkdir(folder + "/XAO")

#
    ------------------------------------------------------------------------------------------------------------
# TUNNEL CREATION
#
    ------------------------------------------------------------------------------------------------------------
model.begin()
partSet = model.moduleDocument()
Part_1 = model.addPart(partSet)
Part_1.setName("Tunnel")
Part_1.result().setName("Tunnel")
Part_1_doc = Part_1.document()
model.addParameter(Part_1_doc, "L", str(TunnelParam[0]), 'Test Section Length')
model.addParameter(Part_1_doc, "W", str(TunnelParam[1]), 'Test Section Width')
model.addParameter(Part_1_doc, "H", str(TunnelParam[2]), 'Test Section Height')
model.addParameter(Part_1_doc, "W_n", str(TunnelParam[3]), 'Nozzle Width')
model.addParameter(Part_1_doc, "H_n", str(TunnelParam[4]), 'Nozzle Height')
model.addParameter(Part_1_doc, "D_n", str(TunnelParam[5]), 'Nozzle Depth into Test Section')
model.addParameter(Part_1_doc, "W_c", str(TunnelParam[6]), 'Collector Width')
model.addParameter(Part_1_doc, "H_c", str(TunnelParam[7]), 'Collector Height')
model.addParameter(Part_1_doc, "D_c", str(TunnelParam[8]), 'Collector Depth into Test
```

```
    Section')
model.addParameter(Part_1_doc, "Alpha", str(TunnelParam[9]), 'Collector Angle')
model.addParameter(Part_1_doc, "Beta", str(TunnelParam[10]), 'Diffuser Angle')

Point_2 = model.addPoint(Part_1_doc, "L/2", "(W*L)/2", "0")
Point_3 = model.addPoint(Part_1_doc, "-L/2", "-(W*L)/2", "H*L")
Point_4 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)", "1.99*(W_n*L)/2", "1.89*(H_n*L)")
Point_5 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-1.365*L", "-1.99*(W_n*L)/2",
    "-0.89*(H_n*L)")
Point_6 = model.addPoint(Part_1_doc, "L/2", "0.58*(W_c*L)", "1.165*(H_c*L)")
Point_7 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+L",
    "-0.58*(W_c*L)", "-0.165*(H_c*L)")
Point_8 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)", "(W_n*L)/2+0.0175*L",
    "(H_n*L)+0.0175*L")
Point_9 = model.addPoint(Part_1_doc, "-L/2", "-(W_n*L)/2-0.0175*L", "0")
Point_10 = model.addPoint(Part_1_doc, "L/2-(D_c*L)",
    "(W_c*L)/2+(D_c*L)*tan(Alpha*(pi/180))+0.03*L", "0")
Point_11 = model.addPoint(Part_1_doc, "L/2",
    "-(W_c*L)/2-(D_c*L)*tan(Alpha*(pi/180))-0.03*L",
    "(H_c*L)+(D_c*L)*tan(Alpha*(pi/180))+0.03*L")
Point_12 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)", "0", "0")
Point_13 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.077*L", "0", "0")
Point_14 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.531*L", "0", "0")
Point_15 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.423*L", "0", "-0.89*(H_n*L)")
Point_16 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.704*L", "0", "-0.89*(H_n*L)")
Point_17 = model.addPoint(Part_1_doc, "0", "0", "(H_n*L)/2")
Point_18 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)", "(W_n*L)/2", "0")
Point_19 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.077*L", "(W_n*L)/2", "0")
Point_20 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.55*L", "(W_n*L)/2", "0")
Point_21 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.423*L", "0.995*(W_n*L)", "0")
Point_22 = model.addPoint(Part_1_doc, "-L/2+(D_n*L)-0.74*L", "0.995*(W_n*L)", "0")
Point_23 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))",
    "0.58*(W_c*L)", "0")
Point_24 = model.addPoint(Part_1_doc, "L/2", "(W_c*L)/2", "0")
Point_25 = model.addPoint(Part_1_doc, "L/2+(0.15*(H_c*L))/(tan(Beta*(pi/180)))",
    "0.58*(W_c*L)", "0")
Point_26 = model.addPoint(Part_1_doc, "L/2+(0.015*(H_c*L))/(tan(Beta*(pi/180)))",
    "(W_c*L)/2", "0")
Point_27 = model.addPoint(Part_1_doc, "L/2", "0", "0")
Point_28 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))", "0",
    "-0.165*(H_c*L)")
Point_29 = model.addPoint(Part_1_doc, "L/2+(0.15*(H_c*L))/(tan(Beta*(pi/180)))", "0",
    "-0.165*(H_c*L)")
Point_30 = model.addPoint(Part_1_doc, "L/2+(0.015*(H_c*L))/(tan(Beta*(pi/180)))", "0", "0")
Point_31 = model.addPoint(Part_1_doc, "0", "0", "(H_c*L)/2")
```

```
Point_32 = model.addPoint(Part_1_doc, "L/2", "-(W_c*L)/2-0.03*L", "0")
Point_33 = model.addPoint(Part_1_doc, "L/2-(D_c*L)",
    "-(W_c*L)/2-(D_c*L)*tan(Alpha*(pi/180))-0.03*L", "0")
Point_34 = model.addPoint(Part_1_doc, "L/2-(D_c*L)",
    "-(W_c*L)/2-(D_c*L)*tan(Alpha*(pi/180))", "0")
Point_35 = model.addPoint(Part_1_doc, "L/2", "-(W_c*L)/2", "0")
Point_36 = model.addPoint(Part_1_doc, "L/2", "0", "(H_c*L)")
Point_37 = model.addPoint(Part_1_doc, "L/2", "0", "(H_c*L)+0.03*L")
Point_38 = model.addPoint(Part_1_doc, "L/2-(D_c*L)", "0",
    "(H_c*L)+(D_c*L)*tan(Alpha*(pi/180))")
Point_39 = model.addPoint(Part_1_doc, "L/2-(D_c*L)", "0",
    "(H_c*L)+(D_c*L)*tan(Alpha*(pi/180))+0.03*L")
Point_40 = model.addPoint(Part_1_doc, "L/2", "(W_c*L)/2", "(H_c*L)")
Point_41 = model.addPoint(Part_1_doc, "L/2", "-(W_c*L)/2", "(H_c*L)")
Point_42 = model.addPoint(Part_1_doc, "L/2-(D_c*L)",
    "(W_c*L)/2+(D_c*L)*tan(Alpha*(pi/180))", "0")
Point_43 = model.addPoint(Part_1_doc, "-L/2+D_n*L-0.01*L", "-W_n*L/2-L*0.0175", "0")
Point_44 = model.addPoint(Part_1_doc, "-L/2+D_n*L+L*0.015", "W_n*L/2+L*0.0175",
    "H_n*L+L*0.0175")
Point_45 = model.addPoint(Part_1_doc, "-L/2+D_n*L+L*0.05", "W_n*L/2+0.0175*L",
    "H_n*L+0.0175*L")
Point_46 = model.addPoint(Part_1_doc, "-L/2+D_n*L+L*0.1", "W_n*L/2+0.0175*L",
    "H_n*L+0.0175*L")
Point_47 = model.addPoint(Part_1_doc, "-L/2+D_n*L-0.01*L", "W_n*L/2-0.006*L",
    "H_n*L-0.006*L")
Point_48 = model.addPoint(Part_1_doc, "0", "0", "H_n*L+0.0175*L")
Point_49 = model.addPoint(Part_1_doc, "-L/2+D_n*L+0.1*L", "-W_n*L/2+0.004*L", "0")
Point_50 = model.addPoint(Part_1_doc, "-L/2+D_n*L-0.01*L", "W_n*L/2+0.004*L",
    "H_n*L+0.004*L")
Point_51 = model.addPoint(Part_1_doc, "-L/2+D_n*L-0.74*L", "-W*L/2", "-0.89*H_n*L")
Point_52 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))", "W*L/2",
    "H*L")
Point_53 = model.addPoint(Part_1_doc, "-L/2+D_n*L-0.37*L",
    "-(W_c*L)/2-(D_c*L)*tan(Alpha*(pi/180))-0.05*L", "-0.89*H_n*L")
Point_54 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))/2",
    "(W_c*L)/2+(D_c*L)*tan(Alpha*(pi/180))+0.05*L",
    "(H_c*L)+(D_c*L)*tan(Alpha*(pi/180))+0.05*L")
Point_55 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+2*L",
    "0.58*W_c*L", "1.165*H_c*L")
Point_56 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+L", "0",
    "-0.165*H_c*L")
Point_57 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.33*L",
    "0", "-0.165*H_c*L")
Point_58 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.33*L",
    "0", "0")
```

```
Point_59 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.67*L",
    "0", "0")
Point_60 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+2*L", "0",
    "0")
Point_61 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+L",
    "-0.58*W_c*L", "0")
Point_62 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.33*L",
    "-0.58*W_c*L", "0")
Point_63 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.33*L",
    "-0.5*W_c*L", "0")
Point_64 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+1.67*L",
    "-0.5*W_c*L", "0")
Point_65 = model.addPoint(Part_1_doc, "L/2+(0.165*(H_c*L))/(tan(Beta*(pi/180)))+2*L",
    "-0.5*W_c*L", "0")

Box_1 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_1"),
    model.selection("VERTEX", "all-in-Point_2"))
Box_2 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_3"),
    model.selection("VERTEX", "all-in-Point_4"))
Box_3 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_5"),
    model.selection("VERTEX", "all-in-Point_6"))
Box_4 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_7"),
    model.selection("VERTEX", "all-in-Point_8"))
Box_5 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_9"),
    model.selection("VERTEX", "all-in-Point_10"))
Box_6 = model.addBox(Part_1_doc, model.selection("VERTEX", "all-in-Point_9"),
    model.selection("VERTEX", "all-in-Point_10"))
Box_7 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_43"),
    model.selection("VERTEX", "Point_42"))
Box_8 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_44"),
    model.selection("VERTEX", "Point_42"))
Box_9 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_45"),
    model.selection("VERTEX", "Point_42"))
Box_10 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_46"),
    model.selection("VERTEX", "Point_48"))
Box_11 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_50"),
    model.selection("VERTEX", "Point_51"))
Box_11.result().setTransparency(0.8)
Box_12 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_52"),
    model.selection("VERTEX", "Point_53"))
Box_12.result().setTransparency(0.8)
Box_13 = model.addBox(Part_1_doc, model.selection("VERTEX", "Point_6"),
    model.selection("VERTEX", "Point_54"))

Axis_4 = model.addAxis(Part_1_doc, model.selection("FACE", "PartSet/YOZ"),
```

```
    model.selection("VERTEX", "Point_16"))
Axis_5 = model.addAxis(Part_1_doc, model.selection("FACE", "PartSet/YOZ"),
    model.selection("VERTEX", "Point_30"))
Axis_6 = model.addAxis(Part_1_doc, model.selection("FACE", "PartSet/YOZ"),
    model.selection("VERTEX", "Point_47"))

Plane_4 = model.addPlane(Part_1_doc, model.selection("VERTEX", "Point_41"),
    model.selection("VERTEX", "Point_23"), model.selection("VERTEX", "Point_39"))
Plane_5 = model.addPlane(Part_1_doc, model.selection("VERTEX", "Point_37"),
    model.selection("VERTEX", "Point_35"), model.selection("VERTEX", "Point_40"))
Plane_6 = model.addPlane(Part_1_doc, model.selection("VERTEX", "Point_33"),
    model.selection("VERTEX", "Point_34"), model.selection("VERTEX", "Point_40"))

Sketch_1 = model.addSketch(Part_1_doc, model.defaultPlane("XOZ"))
SketchBSpline_1 = Sketch_1.addSpline(poles = [(-10.538762, 0), (-19.809442, 0),
    (-17.604082, -3.46937842), (-23.342102, -3.46937842)])
[SketchPoint_1, SketchPoint_2, SketchPoint_3, SketchPoint_4] =
    SketchBSpline_1.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_1, SketchLine_2, SketchLine_3] = SketchBSpline_1.controlPolygon(auxiliary = [0,
    1, 2])
SketchProjection_1 = Sketch_1.addProjection(model.selection("VERTEX", "Point_12"), False)
SketchPoint_5 = SketchProjection_1.createdFeature()
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_5).coordinates(),
    SketchAPI_Point(SketchPoint_1).coordinates())
SketchProjection_2 = Sketch_1.addProjection(model.selection("VERTEX", "Point_13"), False)
SketchPoint_6 = SketchProjection_2.createdFeature()
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_6).coordinates(),
    SketchAPI_Point(SketchPoint_2).coordinates())
SketchProjection_3 = Sketch_1.addProjection(model.selection("VERTEX", "Point_14"), False)
SketchPoint_7 = SketchProjection_3.createdFeature()
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_7).coordinates(),
    SketchAPI_Point(SketchPoint_3).coordinates())
SketchProjection_4 = Sketch_1.addProjection(model.selection("VERTEX", "Point_15"), False)
SketchPoint_8 = SketchProjection_4.createdFeature()
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_8).coordinates(),
    SketchAPI_Point(SketchPoint_4).coordinates())
SketchLine_4 = Sketch_1.addLine(-10.538762, 0, -8.966422000000001, 0)
Sketch_1.setCoincident(SketchBSpline_1.startPoint(), SketchLine_4.startPoint())
SketchProjection_5 = Sketch_1.addProjection(model.selection("VERTEX", "Point_11"), False)
SketchPoint_9 = SketchProjection_5.createdFeature()
Sketch_1.setCoincident(SketchLine_4.endPoint(), SketchPoint_9.result())
SketchLine_5 = Sketch_1.addLine(-8.966422000000001, 0, -8.966422000000001, -3.46937842)
Sketch_1.setCoincident(SketchLine_4.endPoint(), SketchLine_5.startPoint())
Sketch_1.setVertical(SketchLine_5.result())
SketchLine_6 = Sketch_1.addLine(-8.966422000000001, -3.46937842, -23.342102, -3.46937842)
```

```
Sketch_1.setCoincident(SketchLine_5.endPoint(), SketchLine_6.startPoint())
Sketch_1.setCoincident(SketchBSpline_1.endPoint(), SketchLine_6.endPoint())
Sketch_1.setHorizontal(SketchLine_6.result())
SketchProjection_6 = Sketch_1.addProjection(model.selection("EDGE", "Axis_1"), False)
SketchLine_7 = SketchProjection_6.createdFeature()
SketchConstraintMirror_1_objects = [SketchLine_5.result(), SketchLine_4.result(),
    SketchLine_2.result(), SketchLine_1.result(), SketchLine_6.result(),
    SketchLine_3.result()]
SketchConstraintMirror_1 = Sketch_1.addMirror(SketchLine_7.result(),
    SketchConstraintMirror_1_objects)
[SketchLine_8, SketchLine_9, SketchLine_10, SketchLine_11, SketchLine_12, SketchLine_13] =
    SketchConstraintMirror_1.mirrored()
SketchBSpline_2 = Sketch_1.addSpline(poles = [(-10.53876200000001, 3.898178), (-19.809442,
    3.898178), (-17.60408200000001, 7.36755642), (-23.34210200000002, 7.36755642)])
[SketchPoint_10, SketchPoint_11, SketchPoint_12, SketchPoint_13] =
    SketchBSpline_2.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_14, SketchLine_15, SketchLine_16] = SketchBSpline_2.controlPolygon(auxiliary =
    [0, 1, 2])
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_10).coordinates(),
    SketchAPI_Line(SketchLine_11).startPoint())
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_11).coordinates(),
    SketchAPI_Line(SketchLine_10).startPoint())
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_12).coordinates(),
    SketchAPI_Line(SketchLine_13).startPoint())
Sketch_1.setCoincident(SketchAPI_Point(SketchPoint_13).coordinates(),
    SketchAPI_Line(SketchLine_13).endPoint())
model.do()


Sketch_2 = model.addSketch(Part_1_doc, model.defaultPlane("XOY"))
SketchBSpline_3 = Sketch_2.addSpline(poles = [(-10.538762, 3.470379), (-20.197422,
    3.470379), (-17.604082, 6.90605421), (-24.077222, 6.90605421)])
[SketchPoint_14, SketchPoint_15, SketchPoint_16, SketchPoint_17] =
    SketchBSpline_3.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_17, SketchLine_18, SketchLine_19] = SketchBSpline_3.controlPolygon(auxiliary =
    [0, 1, 2])
SketchProjection_7 = Sketch_2.addProjection(model.selection("VERTEX", "Point_18"), False)
SketchPoint_18 = SketchProjection_7.createdFeature()
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_18).coordinates(),
    SketchAPI_Point(SketchPoint_14).coordinates())
SketchProjection_8 = Sketch_2.addProjection(model.selection("VERTEX", "Point_19"), False)
SketchPoint_19 = SketchProjection_8.createdFeature()
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_19).coordinates(),
    SketchAPI_Point(SketchPoint_15).coordinates())
SketchProjection_9 = Sketch_2.addProjection(model.selection("VERTEX", "Point_20"), False)
```

```
SketchPoint_20 = SketchProjection_9.createdFeature()
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_20).coordinates(),
    SketchAPI_Point(SketchPoint_16).coordinates())
SketchProjection_10 = Sketch_2.addProjection(model.selection("VERTEX", "Point_21"), False)
SketchPoint_21 = SketchProjection_10.createdFeature()
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_21).coordinates(),
    SketchAPI_Point(SketchPoint_17).coordinates())
SketchLine_20 = Sketch_2.addLine(-10.538762, 3.470379, -8.966422000000001, 3.470379)
Sketch_2.setCoincident(SketchBSpline_3.startPoint(), SketchLine_20.startPoint())
SketchProjection_11 = Sketch_2.addProjection(model.selection("VERTEX", "Point_17"), False)
SketchPoint_22 = SketchProjection_11.createdFeature()
Sketch_2.setCoincident(SketchLine_20.endPoint(), SketchPoint_22.result())
SketchLine_21 = Sketch_2.addLine(-8.966422000000001, 3.470379, -8.966422000000001,
    6.90605421)
Sketch_2.setCoincident(SketchLine_20.endPoint(), SketchLine_21.startPoint())
Sketch_2.setVertical(SketchLine_21.result())
SketchLine_22 = Sketch_2.addLine(-8.966422000000001, 6.90605421, -24.077222, 6.90605421)
Sketch_2.setCoincident(SketchLine_21.endPoint(), SketchLine_22.startPoint())
Sketch_2.setCoincident(SketchBSpline_3.endPoint(), SketchLine_22.endPoint())
Sketch_2.setHorizontal(SketchLine_22.result())
SketchProjection_12 = Sketch_2.addProjection(model.selection("EDGE", "PartSet/OX"), False)
SketchLine_23 = SketchProjection_12.createdFeature()
SketchConstraintMirror_2_objects = [SketchLine_17.result(), SketchLine_20.result(),
    SketchLine_19.result(), SketchLine_22.result(), SketchLine_18.result(),
    SketchLine_21.result()]
SketchConstraintMirror_2 = Sketch_2.addMirror(SketchLine_23.result(),
    SketchConstraintMirror_2_objects)
[SketchLine_24, SketchLine_25, SketchLine_26, SketchLine_27, SketchLine_28, SketchLine_29]
    = SketchConstraintMirror_2.mirrored()
SketchBSpline_4 = Sketch_2.addSpline(poles = [(-10.538762, -3.470379), (-20.197422,
    -3.470379), (-17.604082, -6.90605421), (-24.077222, -6.90605421)])
[SketchPoint_23, SketchPoint_24, SketchPoint_25, SketchPoint_26] =
    SketchBSpline_4.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_30, SketchLine_31, SketchLine_32] = SketchBSpline_4.controlPolygon(auxiliary =
    [0, 1, 2])
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_23).coordinates(),
    SketchAPI_Line(SketchLine_24).startPoint())
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_24).coordinates(),
    SketchAPI_Line(SketchLine_28).startPoint())
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_25).coordinates(),
    SketchAPI_Line(SketchLine_26).startPoint())
Sketch_2.setCoincident(SketchAPI_Point(SketchPoint_26).coordinates(),
    SketchAPI_Line(SketchLine_26).endPoint())
model.do()
```

```
Sketch_3 = model.addSketch(Part_1_doc, model.standardPlane("XOY"))
SketchBSpline_5 = Sketch_3.addSpline(poles = [(10.21, 4.347418), (12.07399799405463,
    4.347418), (28.84997994054626, 5.04300488), (30.71397793460088, 5.04300488)])
[SketchPoint_27, SketchPoint_28, SketchPoint_29, SketchPoint_30] =
    SketchBSpline_5.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_33, SketchLine_34, SketchLine_35] = SketchBSpline_5.controlPolygon(auxiliary =
    [0, 1, 2])
SketchProjection_13 = Sketch_3.addProjection(model.selection("VERTEX", "Point_23"), False)
SketchPoint_31 = SketchProjection_13.createdFeature()
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_31).coordinates(),
    SketchAPI_Point(SketchPoint_27).coordinates())
SketchProjection_14 = Sketch_3.addProjection(model.selection("VERTEX", "Point_25"), False)
SketchPoint_32 = SketchProjection_14.createdFeature()
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_32).coordinates(),
    SketchAPI_Point(SketchPoint_28).coordinates())
SketchProjection_15 = Sketch_3.addProjection(model.selection("VERTEX", "Point_24"), False)
SketchPoint_33 = SketchProjection_15.createdFeature()
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_29).coordinates(),
    SketchAPI_Point(SketchPoint_33).coordinates())
SketchProjection_16 = Sketch_3.addProjection(model.selection("VERTEX", "Point_22"), False)
SketchPoint_34 = SketchProjection_16.createdFeature()
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_30).coordinates(),
    SketchAPI_Point(SketchPoint_34).coordinates())
SketchLine_36 = Sketch_3.addLine(10.21, 4.347418, 10.21, 5.04300488)
Sketch_3.setCoincident(SketchBSpline_5.startPoint(), SketchLine_36.startPoint())
SketchLine_37 = Sketch_3.addLine(10.21, 5.04300488, 30.71397793460088, 5.04300488)
Sketch_3.setCoincident(SketchLine_36.endPoint(), SketchLine_37.startPoint())
Sketch_3.setCoincident(SketchBSpline_5.endPoint(), SketchLine_37.endPoint())
Sketch_3.setHorizontal(SketchLine_37.result())
Sketch_3.setVertical(SketchLine_36.result())
SketchProjection_17 = Sketch_3.addProjection(model.selection("EDGE", "PartSet/OX"), False)
SketchLine_38 = SketchProjection_17.createdFeature()
SketchConstraintMirror_3_objects = [SketchLine_36.result(), SketchLine_33.result(),
    SketchLine_34.result(), SketchLine_37.result(), SketchLine_35.result()]
SketchConstraintMirror_3 = Sketch_3.addMirror(SketchLine_38.result(),
    SketchConstraintMirror_3_objects)
[SketchLine_39, SketchLine_40, SketchLine_41, SketchLine_42, SketchLine_43] =
    SketchConstraintMirror_3.mirrored()
SketchBSpline_6 = Sketch_3.addSpline(poles = [(10.21, -4.347418), (12.07399799405463,
    -4.347418), (28.84997994054626, -5.04300488), (30.71397793460088, -5.04300488)])
[SketchPoint_35, SketchPoint_36, SketchPoint_37, SketchPoint_38] =
    SketchBSpline_6.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_44, SketchLine_45, SketchLine_46] = SketchBSpline_6.controlPolygon(auxiliary =
    [0, 1, 2])
```

```
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_35).coordinates(),
    SketchAPI_Line(SketchLine_39).startPoint())
Sketch_3.setCoincident(SketchAPI_Line(SketchLine_45).startPoint(),
    SketchAPI_Line(SketchLine_40).endPoint())
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_37).coordinates(),
    SketchAPI_Line(SketchLine_41).endPoint())
Sketch_3.setCoincident(SketchAPI_Point(SketchPoint_38).coordinates(),
    SketchAPI_Line(SketchLine_42).endPoint())
model.do()


Sketch_4 = model.addSketch(Part_1_doc, model.standardPlane("XOZ"))
SketchBSpline_7 = Sketch_4.addSpline(poles = [(10.21, 0), (12.07399799405463, 0),
    (28.84997994054626, -0.8952230100000005), (30.71397793460088, -0.8952230100000005)])
[SketchPoint_39, SketchPoint_40, SketchPoint_41, SketchPoint_42] =
    SketchBSpline_7.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_47, SketchLine_48, SketchLine_49] = SketchBSpline_7.controlPolygon(auxiliary =
    [0, 1, 2])
SketchProjection_18 = Sketch_4.addProjection(model.selection("VERTEX", "Point_26"), False)
SketchPoint_43 = SketchProjection_18.createdFeature()
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_43).coordinates(),
    SketchAPI_Point(SketchPoint_39).coordinates())
SketchProjection_19 = Sketch_4.addProjection(model.selection("VERTEX", "Point_29"), False)
SketchPoint_44 = SketchProjection_19.createdFeature()
Sketch_4.setCoincident(SketchAPI_Line(SketchLine_48).startPoint(),
    SketchAPI_Point(SketchPoint_44).coordinates())
SketchProjection_20 = Sketch_4.addProjection(model.selection("VERTEX", "Point_28"), False)
SketchPoint_45 = SketchProjection_20.createdFeature()
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_41).coordinates(),
    SketchAPI_Point(SketchPoint_45).coordinates())
SketchProjection_21 = Sketch_4.addProjection(model.selection("VERTEX", "Point_27"), False)
SketchPoint_46 = SketchProjection_21.createdFeature()
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_42).coordinates(),
    SketchAPI_Point(SketchPoint_46).coordinates())
SketchLine_50 = Sketch_4.addLine(10.21, 0, 10.21, -0.89522301)
Sketch_4.setCoincident(SketchBSpline_7.startPoint(), SketchLine_50.startPoint())
Sketch_4.setVertical(SketchLine_50.result())
SketchLine_51 = Sketch_4.addLine(10.21, -0.89522301, 30.71397793460088, -0.8952230100000005)
Sketch_4.setCoincident(SketchLine_50.endPoint(), SketchLine_51.startPoint())
Sketch_4.setCoincident(SketchBSpline_7.endPoint(), SketchLine_51.endPoint())
Sketch_4.setHorizontal(SketchLine_51.result())
SketchProjection_22 = Sketch_4.addProjection(model.selection("EDGE", "Axis_2"), False)
SketchLine_52 = SketchProjection_22.createdFeature()
SketchConstraintMirror_4_objects = [SketchLine_50.result(), SketchLine_48.result(),
    SketchLine_51.result(), SketchLine_49.result(), SketchLine_47.result()]
```

```
SketchConstraintMirror_4 = Sketch_4.addMirror(SketchLine_52.result(),
    SketchConstraintMirror_4_objects)
[SketchLine_53, SketchLine_54, SketchLine_55, SketchLine_56, SketchLine_57] =
    SketchConstraintMirror_4.mirrored()
SketchBSpline_8 = Sketch_4.addSpline(poles = [(10.21000000000002, 5.425594),
    (12.07399799405462, 5.425594), (28.84997994054627, 6.320817010000001),
    (30.71397793460087, 6.320817010000001)])
[SketchPoint_47, SketchPoint_48, SketchPoint_49, SketchPoint_50] =
    SketchBSpline_8.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_58, SketchLine_59, SketchLine_60] = SketchBSpline_8.controlPolygon(auxiliary =
    [0, 1, 2])
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_47).coordinates(),
    SketchAPI_Line(SketchLine_53).startPoint())
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_48).coordinates(),
    SketchAPI_Line(SketchLine_57).endPoint())
Sketch_4.setCoincident(SketchAPI_Line(SketchLine_59).endPoint(),
    SketchAPI_Line(SketchLine_54).endPoint())
Sketch_4.setCoincident(SketchAPI_Point(SketchPoint_50).coordinates(),
    SketchAPI_Line(SketchLine_55).endPoint())
model.do()


Sketch_5 = model.addSketch(Part_1_doc, model.standardPlane("XOY"))
SketchLine_61 = Sketch_5.addLine(10.21, -4.960018, 10.21, -4.347418)
SketchProjection_23 = Sketch_5.addProjection(model.selection("VERTEX", "Point_31"), False)
SketchPoint_51 = SketchProjection_23.createdFeature()
Sketch_5.setCoincident(SketchLine_61.startPoint(), SketchPoint_51.result())
SketchProjection_24 = Sketch_5.addProjection(model.selection("VERTEX", "Point_34"), False)
SketchPoint_52 = SketchProjection_24.createdFeature()
Sketch_5.setCoincident(SketchLine_61.endPoint(), SketchPoint_52.result())
SketchLine_62 = Sketch_5.addLine(10.21, -4.347418, 6.375124, -5.374969927273495)
Sketch_5.setCoincident(SketchLine_61.endPoint(), SketchLine_62.startPoint())
SketchProjection_25 = Sketch_5.addProjection(model.selection("VERTEX", "Point_33"), False)
SketchPoint_53 = SketchProjection_25.createdFeature()
Sketch_5.setCoincident(SketchLine_62.endPoint(), SketchPoint_53.result())
SketchLine_63 = Sketch_5.addLine(6.375124, -5.374969927273495, 6.375124, -5.987569927273494)
Sketch_5.setCoincident(SketchLine_62.endPoint(), SketchLine_63.startPoint())
SketchProjection_26 = Sketch_5.addProjection(model.selection("VERTEX", "Point_32"), False)
SketchPoint_54 = SketchProjection_26.createdFeature()
Sketch_5.setCoincident(SketchLine_63.endPoint(), SketchPoint_54.result())
SketchLine_64 = Sketch_5.addLine(6.375124, -5.987569927273494, 10.21, -4.960018)
Sketch_5.setCoincident(SketchLine_63.endPoint(), SketchLine_64.startPoint())
Sketch_5.setCoincident(SketchLine_61.startPoint(), SketchLine_64.endPoint())
SketchProjection_27 = Sketch_5.addProjection(model.selection("EDGE", "PartSet/OX"), False)
SketchLine_65 = SketchProjection_27.createdFeature()
```

```
SketchConstraintMirror_5_objects = [SketchLine_64.result(), SketchLine_61.result(),
    SketchLine_63.result(), SketchLine_62.result()]
SketchConstraintMirror_5 = Sketch_5.addMirror(SketchLine_65.result(),
    SketchConstraintMirror_5_objects)
[SketchLine_66, SketchLine_67, SketchLine_68, SketchLine_69] =
    SketchConstraintMirror_5.mirrored()
model.do()


Sketch_6 = model.addSketch(Part_1_doc, model.defaultPlane("XOZ"))
SketchLine_70 = Sketch_6.addLine(10.21, 5.425594, 10.21, 6.038194000000001)
SketchProjection_28 = Sketch_6.addProjection(model.selection("VERTEX", "Point_35"), False)
SketchPoint_55 = SketchProjection_28.createdFeature()
Sketch_6.setCoincident(SketchLine_70.startPoint(), SketchPoint_55.result())
SketchProjection_29 = Sketch_6.addProjection(model.selection("VERTEX", "Point_36"), False)
SketchPoint_56 = SketchProjection_29.createdFeature()
Sketch_6.setCoincident(SketchLine_70.endPoint(), SketchPoint_56.result())
SketchLine_71 = Sketch_6.addLine(10.21, 6.038194000000001, 6.375124, 7.065745927273495)
Sketch_6.setCoincident(SketchLine_70.endPoint(), SketchLine_71.startPoint())
SketchProjection_30 = Sketch_6.addProjection(model.selection("VERTEX", "Point_38"), False)
SketchPoint_57 = SketchProjection_30.createdFeature()
Sketch_6.setCoincident(SketchLine_71.endPoint(), SketchPoint_57.result())
SketchLine_72 = Sketch_6.addLine(6.375124, 7.065745927273495, 6.375124, 6.453145927273495)
Sketch_6.setCoincident(SketchLine_71.endPoint(), SketchLine_72.startPoint())
SketchProjection_31 = Sketch_6.addProjection(model.selection("VERTEX", "Point_37"), False)
SketchPoint_58 = SketchProjection_31.createdFeature()
Sketch_6.setCoincident(SketchLine_72.endPoint(), SketchPoint_58.result())
SketchLine_73 = Sketch_6.addLine(6.375124, 6.453145927273495, 10.21, 5.425594)
Sketch_6.setCoincident(SketchLine_72.endPoint(), SketchLine_73.startPoint())
Sketch_6.setCoincident(SketchLine_70.startPoint(), SketchLine_73.endPoint())
model.do()


Sketch_7 = model.addSketch(Part_1_doc, model.selection("FACE", "Box_9_1/Top"))
SketchLine_74 = Sketch_7.addLine(-6.924422000000002, 3.827729, -9.170622000000002, 3.827729)
SketchProjection_32 = Sketch_7.addProjection(model.selection("VERTEX", "Point_45"), False)
SketchPoint_59 = SketchProjection_32.createdFeature()
Sketch_7.setCoincident(SketchLine_74.startPoint(), SketchPoint_59.result())
Sketch_7.setHorizontal(SketchLine_74.result())
SketchLine_75 = Sketch_7.addLine(-9.170622000000002, 3.827729, -9.170622000000002, 3.552059)
Sketch_7.setCoincident(SketchLine_74.endPoint(), SketchLine_75.startPoint())
Sketch_7.setVertical(SketchLine_75.result())
SketchLine_76 = Sketch_7.addLine(-9.170622000000002, 3.552059, -6.924422000000002, 3.827729)
Sketch_7.setCoincident(SketchLine_75.endPoint(), SketchLine_76.startPoint())
Sketch_7.setCoincident(SketchLine_74.startPoint(), SketchLine_76.endPoint())
```

```
SketchProjection_33 = Sketch_7.addProjection(model.selection("VERTEX", "Point_49"), True)
SketchPoint_60 = SketchProjection_33.createdFeature()
Sketch_7.setCoincident(SketchLine_75.endPoint(),
    SketchAPI_Point(SketchPoint_60).coordinates())
SketchProjection_34 = Sketch_7.addProjection(model.selection("EDGE", "Axis_3"), False)
SketchProjection_34.setName("SketchProjection_36")
SketchProjection_34.result().setName("SketchProjection_36")
SketchLine_77 = SketchProjection_34.createdFeature()
SketchLine_77.setName("SketchLine_80")
SketchLine_77.result().setName("SketchLine_80")
SketchConstraintMirror_6_objects = [SketchLine_76.result(), SketchLine_74.result(),
    SketchLine_75.result()]
SketchConstraintMirror_6 = Sketch_7.addMirror(SketchLine_77.result(),
    SketchConstraintMirror_6_objects)
[SketchLine_78, SketchLine_79, SketchLine_80] = SketchConstraintMirror_6.mirrored()
SketchLine_80.setName("SketchLine_83")
SketchLine_80.result().setName("SketchLine_83")
SketchLine_79.setName("SketchLine_82")
SketchLine_79.result().setName("SketchLine_82")
SketchLine_78.setName("SketchLine_81")
SketchLine_78.result().setName("SketchLine_81")
model.do()


Sketch_8 = model.addSketch(Part_1_doc, model.selection("FACE", "Box_9_1/Right"))
SketchLine_81 = Sketch_8.addLine(-6.924422000000002, -4.255528, -9.170622000000002,
    -4.255528)
SketchLine_81.setName("SketchLine_77")
SketchLine_81.result().setName("SketchLine_77")
SketchProjection_35 = Sketch_8.addProjection(model.selection("VERTEX", "Point_45"), False)
SketchProjection_35.setName("SketchProjection_34")
SketchProjection_35.result().setName("SketchProjection_34")
SketchPoint_61 = SketchProjection_35.createdFeature()
Sketch_8.setCoincident(SketchLine_81.startPoint(), SketchPoint_61.result())
Sketch_8.setHorizontal(SketchLine_81.result())
SketchLine_82 = Sketch_8.addLine(-9.170622000000002, -4.255528, -9.170622000000002,
    -3.979858)
SketchLine_82.setName("SketchLine_78")
SketchLine_82.result().setName("SketchLine_78")
Sketch_8.setCoincident(SketchLine_81.endPoint(), SketchLine_82.startPoint())
Sketch_8.setVertical(SketchLine_82.result())
SketchLine_83 = Sketch_8.addLine(-9.170622000000002, -3.979858, -6.924422000000002,
    -4.255528)
SketchLine_83.setName("SketchLine_79")
SketchLine_83.result().setName("SketchLine_79")
```

```
Sketch_8.setCoincident(SketchLine_82.endPoint(), SketchLine_83.startPoint())
Sketch_8.setCoincident(SketchLine_81.startPoint(), SketchLine_83.endPoint())
SketchProjection_36 = Sketch_8.addProjection(model.selection("VERTEX", "Point_49"), True)
SketchProjection_36.setName("SketchProjection_35")
SketchProjection_36.result().setName("SketchProjection_35")
SketchPoint_62 = SketchProjection_36.createdFeature()
Sketch_8.setCoincident(SketchLine_82.endPoint(),
    SketchAPI_Point(SketchPoint_62).coordinates())
model.do()


Sketch_9 = model.addSketch(Part_1_doc, model.defaultPlane("XOZ"))
SketchProjection_37 = Sketch_9.addProjection(model.selection("VERTEX", "Point_55"), False)
SketchPoint_63 = SketchProjection_37.createdFeature()
SketchProjection_38 = Sketch_9.addProjection(model.selection("VERTEX", "Point_56"), False)
SketchPoint_64 = SketchProjection_38.createdFeature()
SketchProjection_39 = Sketch_9.addProjection(model.selection("VERTEX", "Point_57"), False)
SketchPoint_65 = SketchProjection_39.createdFeature()
SketchProjection_40 = Sketch_9.addProjection(model.selection("VERTEX", "Point_58"), False)
SketchPoint_66 = SketchProjection_40.createdFeature()
SketchBSpline_9 = Sketch_9.addSpline(poles = [(51.13397793460088, -0.89522301),
    (57.87257793460088, -0.89522301), (57.87257793460088, 0), (64.81537793460087, 0)])
[SketchPoint_67, SketchPoint_68, SketchPoint_69, SketchPoint_70] =
    SketchBSpline_9.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_84, SketchLine_85, SketchLine_86] = SketchBSpline_9.controlPolygon(auxiliary =
    [0, 1, 2])
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_67).coordinates(),
    SketchPoint_63.result())
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_68).coordinates(),
    SketchPoint_64.result())
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_69).coordinates(),
    SketchPoint_65.result())
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_70).coordinates(),
    SketchPoint_66.result())
SketchLine_87 = Sketch_9.addLine(71.55397793460088, 0, 64.81537793460087, 0)
SketchProjection_41 = Sketch_9.addProjection(model.selection("VERTEX", "Point_59"), False)
SketchPoint_71 = SketchProjection_41.createdFeature()
Sketch_9.setCoincident(SketchLine_87.startPoint(), SketchPoint_71.result())
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_66).coordinates(),
    SketchLine_87.endPoint())
SketchLine_88 = Sketch_9.addLine(51.13397793460088, -0.89522301, 71.55397793460088,
    -0.89522301)
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_63).coordinates(),
    SketchLine_88.startPoint())
Sketch_9.setHorizontal(SketchLine_88.result())
```

```
SketchLine_89 = Sketch_9.addLine(71.55397793460088, -0.89522301, 71.55397793460088, 0)
Sketch_9.setCoincident(SketchLine_88.endPoint(), SketchLine_89.startPoint())
Sketch_9.setCoincident(SketchLine_87.startPoint(), SketchLine_89.endPoint())
Sketch_9.setVertical(SketchLine_89.result())
SketchProjection_42 = Sketch_9.addProjection(model.selection("EDGE", "Axis_2"), False)
SketchLine_90 = SketchProjection_42.createdFeature()
SketchConstraintMirror_7_objects = [SketchLine_85.result(), SketchLine_84.result(),
    SketchLine_89.result(), SketchLine_88.result(), SketchLine_87.result(),
    SketchLine_86.result()]
SketchConstraintMirror_7 = Sketch_9.addMirror(SketchLine_90.result(),
    SketchConstraintMirror_7_objects)
[SketchLine_91, SketchLine_92, SketchLine_93, SketchLine_94, SketchLine_95, SketchLine_96]
    = SketchConstraintMirror_7.mirrored()
SketchBSpline_10 = Sketch_9.addSpline(poles = [(51.13397793460088, 6.320817010000001),
    (57.87257793460088, 6.320817010000001), (57.87257793460088, 5.425594),
    (64.81537793460087, 5.425594)])
[SketchPoint_72, SketchPoint_73, SketchPoint_74, SketchPoint_75] =
    SketchBSpline_10.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_97, SketchLine_98, SketchLine_99] = SketchBSpline_10.controlPolygon(auxiliary =
    [0, 1, 2])
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_72).coordinates(),
    SketchAPI_Line(SketchLine_92).startPoint())
Sketch_9.setCoincident(SketchAPI_Line(SketchLine_97).endPoint(),
    SketchAPI_Line(SketchLine_92).endPoint())
Sketch_9.setCoincident(SketchAPI_Line(SketchLine_98).endPoint(),
    SketchAPI_Line(SketchLine_91).endPoint())
Sketch_9.setCoincident(SketchAPI_Point(SketchPoint_75).coordinates(),
    SketchAPI_Line(SketchLine_96).endPoint())
model.do()


Sketch_10 = model.addSketch(Part_1_doc, model.defaultPlane("XOY"))
SketchBSpline_11 = Sketch_10.addSpline(poles = [(51.13397793460088, -5.04300488),
    (57.87257793460088, -5.04300488), (57.87257793460088, -4.347418), (64.81537793460087,
    -4.347418)])
[SketchPoint_76, SketchPoint_77, SketchPoint_78, SketchPoint_79] =
    SketchBSpline_11.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_100, SketchLine_101, SketchLine_102] =
    SketchBSpline_11.controlPolygon(auxiliary = [0, 1, 2])
SketchProjection_43 = Sketch_10.addProjection(model.selection("VERTEX", "Point_60"), False)
SketchPoint_80 = SketchProjection_43.createdFeature()
Sketch_10.setCoincident(SketchAPI_Point(SketchPoint_80).coordinates(),
    SketchAPI_Point(SketchPoint_76).coordinates())
SketchProjection_44 = Sketch_10.addProjection(model.selection("VERTEX", "Point_61"), False)
SketchPoint_81 = SketchProjection_44.createdFeature()
```

```
Sketch_10.setCoincident(SketchAPI_Point(SketchPoint_77).coordinates(),
    SketchAPI_Point(SketchPoint_81).coordinates())
SketchProjection_45 = Sketch_10.addProjection(model.selection("VERTEX", "Point_62"), False)
SketchPoint_82 = SketchProjection_45.createdFeature()
Sketch_10.setCoincident(SketchAPI_Line(SketchLine_101).endPoint(),
    SketchAPI_Point(SketchPoint_82).coordinates())
SketchProjection_46 = Sketch_10.addProjection(model.selection("VERTEX", "Point_63"), False)
SketchPoint_83 = SketchProjection_46.createdFeature()
Sketch_10.setCoincident(SketchAPI_Point(SketchPoint_79).coordinates(),
    SketchAPI_Point(SketchPoint_83).coordinates())
SketchLine_103 = Sketch_10.addLine(64.81537793460087, -4.347418, 71.55397793460088,
    -4.347418)
Sketch_10.setCoincident(SketchBSpline_11.endPoint(), SketchLine_103.startPoint())
SketchProjection_47 = Sketch_10.addProjection(model.selection("VERTEX", "Point_64"), False)
SketchPoint_84 = SketchProjection_47.createdFeature()
Sketch_10.setCoincident(SketchLine_103.endPoint(), SketchPoint_84.result())
SketchLine_104 = Sketch_10.addLine(71.55397793460088, -4.347418, 71.55397793460088,
    -5.04300488)
Sketch_10.setCoincident(SketchLine_103.endPoint(), SketchLine_104.startPoint())
Sketch_10.setVertical(SketchLine_104.result())
SketchLine_105 = Sketch_10.addLine(71.55397793460088, -5.04300488, 51.13397793460088,
    -5.04300488)
Sketch_10.setCoincident(SketchLine_104.endPoint(), SketchLine_105.startPoint())
Sketch_10.setCoincident(SketchBSpline_11.startPoint(), SketchLine_105.endPoint())
Sketch_10.setHorizontal(SketchLine_105.result())
SketchProjection_48 = Sketch_10.addProjection(model.selection("EDGE", "PartSet/OX"), False)
SketchLine_106 = SketchProjection_48.createdFeature()
SketchConstraintMirror_8_objects = [SketchLine_105.result(), SketchLine_100.result(),
    SketchLine_101.result(), SketchLine_102.result(), SketchLine_104.result(),
    SketchLine_103.result()]
SketchConstraintMirror_8 = Sketch_10.addMirror(SketchLine_106.result(),
    SketchConstraintMirror_8_objects)
[SketchLine_107, SketchLine_108, SketchLine_109, SketchLine_110, SketchLine_111,
    SketchLine_112] = SketchConstraintMirror_8.mirrored()
SketchBSpline_12 = Sketch_10.addSpline(poles = [(51.13397793460088, 5.04300488),
    (57.87257793460088, 5.04300488), (57.87257793460088, 4.347418), (64.81537793460087,
    4.347418)])
[SketchPoint_85, SketchPoint_86, SketchPoint_87, SketchPoint_88] =
    SketchBSpline_12.controlPoles(auxiliary = [0, 1, 2, 3])
[SketchLine_113, SketchLine_114, SketchLine_115] =
    SketchBSpline_12.controlPolygon(auxiliary = [0, 1, 2])
Sketch_10.setCoincident(SketchAPI_Point(SketchPoint_85).coordinates(),
    SketchAPI_Line(SketchLine_108).startPoint())
Sketch_10.setCoincident(SketchAPI_Line(SketchLine_113).endPoint(),
    SketchAPI_Line(SketchLine_108).endPoint())
```

```
Sketch_10.setCoincident(SketchAPI_Point(SketchPoint_87).coordinates(),
    SketchAPI_Line(SketchLine_109).endPoint())
Sketch_10.setCoincident(SketchBSpline_12.endPoint(),
    SketchAPI_Line(SketchLine_110).endPoint())
model.do()


ExtrusionCut_1 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_5")], model.selection(), model.selection("FACE", "Plane_2"), "0.03*L",
    model.selection(), 0, [model.selection("SOLID", "Box_5_1")])
ExtrusionCut_2 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_6")], model.selection(), model.selection("FACE", "Plane_3"), 0,
    model.selection("FACE", "Plane_1"), 0, [model.selection("SOLID", "ExtrusionCut_1_1")])
ExtrusionCut_3 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_1"), model.selection("COMPOUND", "Sketch_2")], model.selection(),
    [model.selection("SOLID", "Box_2_1")])
ExtrusionCut_4 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_3"), model.selection("COMPOUND", "Sketch_4")], model.selection(),
    [model.selection("SOLID", "Box_3_1")])
Cut_1 = model.addCut(Part_1_doc, [model.selection("SOLID", "Box_1_1")],
    [model.selection("SOLID", "Box_6_1"), model.selection("SOLID", "Box_4_1")],
    keepSubResults = True)
ExtrusionCut_5_objects_2 = [model.selection("SOLID", "Box_7_1"),
                            model.selection("SOLID", "Box_8_1"),
                            model.selection("SOLID", "Box_9_1")]
ExtrusionCut_5 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_7"), model.selection("COMPOUND", "Sketch_8")], model.selection(),
    ExtrusionCut_5_objects_2)
Cut_2_objects_1 = [model.selection("SOLID", "ExtrusionCut_5_1"),
                   model.selection("SOLID", "ExtrusionCut_5_2"),
                   model.selection("SOLID", "ExtrusionCut_5_3")]
Cut_2 = model.addCut(Part_1_doc, Cut_2_objects_1, [model.selection("SOLID", "Box_10_1")],
    keepSubResults = True)
Cut_2.result().setTransparency(0.8)
Cut_2.results()[1].setTransparency(0.8)
Cut_2.results()[2].setTransparency(0.8)
ExtrusionCut_6 = model.addExtrusionCut(Part_1_doc, [model.selection("COMPOUND",
    "Sketch_9"), model.selection("COMPOUND", "Sketch_10")], model.selection(),
    [model.selection("SOLID", "Box_13_1")])
Fuse_1_objects_1 = [model.selection("SOLID", "ExtrusionCut_2_1_1"),
                    model.selection("SOLID", "ExtrusionCut_2_1_2"),
                    model.selection("SOLID", "ExtrusionCut_3_1"),
                    model.selection("SOLID", "ExtrusionCut_4_1"),
                    model.selection("SOLID", "Cut_1_1"),
                    model.selection("SOLID", "ExtrusionCut_6_1")]
```

```
Fuse_1 = model.addFuse(Part_1_doc, Fuse_1_objects_1, removeEdges = True, keepSubResults =
    True)
Fillet_1_objects = [model.selection("FACE", "Fuse_1_1/Modified_Face&Box_6_1/Back"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_5/SketchLine_62"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_5/SketchLine_64"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&ExtrusionCut_1_1/To_Face_1&Sketch_6/SketchLine_71&Extrusio
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_5/SketchLine_69"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_5/SketchLine_66"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_3/SketchBSpline_6"),
                    model.selection("FACE",
                        "Fuse_1_1/Modified_Face&Sketch_3/SketchBSpline_5")]
Fillet_1 = model.addFillet(Part_1_doc, Fillet_1_objects, "0.02*L/2", keepSubResults = True)
Fillet_2_objects = [model.selection("FACE",
    "Fillet_1_1/MF:Fillet&Box_1_1/Bottom&Sketch_1/SketchLine_4&Box_5_1/Bottom"),
                    model.selection("FACE", "Fuse_1_1/Modified_Face&Box_4_1/Front"),
                    model.selection("FACE", "Fuse_1_1/Modified_Face&Box_1_1/Left"),
                    model.selection("FACE",
                        "Fillet_1_1/MF:Fillet&Box_1_1/Front&Box_5_1/Front"),
                    model.selection("FACE", "Fuse_1_1/Modified_Face&Box_1_1/Right"),
                    model.selection("FACE", "Box_4_1/Top"),
                    model.selection("FACE", "Box_4_1/Right"),
                    model.selection("FACE", "Box_4_1/Left"),
                    model.selection("FACE", "Box_1_1/Top"),
                    model.selection("FACE",
                        "ExtrusionCut_3_1/Generated_Face&Sketch_1/SketchBSpline_2")]
Fillet_2 = model.addFillet(Part_1_doc, Fillet_2_objects, "0.008*L/2", keepSubResults = True)
Fillet_2.result().setTransparency(0.8)

Export_1 = model.exportToXAO(Part_1_doc, folder + '/XAO/Tunnel.xao',
    model.selection("SOLID", "Fillet_2_1"), 'XAO')
Export_2 = model.exportToXAO(Part_1_doc, folder + '/XAO/NShear1.xao',
    model.selection("SOLID", "Cut_2_1"), 'XAO')
Export_3 = model.exportToXAO(Part_1_doc, folder + '/XAO/NShear2.xao',
    model.selection("SOLID", "Cut_2_2"), 'XAO')
Export_4 = model.exportToXAO(Part_1_doc, folder + '/XAO/NShear3.xao',
    model.selection("SOLID", "Cut_2_3"), 'XAO')
Export_5 = model.exportToXAO(Part_1_doc, folder + '/XAO/ImportantArea1.xao',
    model.selection("SOLID", "Box_11_1"), 'XAO')
Export_6 = model.exportToXAO(Part_1_doc, folder + '/XAO/ImportantArea2.xao',
```

```
    model.selection("SOLID", "Box_12_1"), 'XAO')

model.do()


#
    ----------------------------------------------------------------------------------------------------
# VEHICLE CREATION
#
    ----------------------------------------------------------------------------------------------------
Part_2 = model.addPart(partSet)
Part_2.setName("Vehicle")
Part_2.result().setName("Vehicle")
Part_2_doc = Part_2.document()
model.addParameter(Part_2_doc, "l", str(VehicleParam[0]), 'Length')
model.addParameter(Part_2_doc, "w", str(VehicleParam[1]), 'Width')
model.addParameter(Part_2_doc, "h", str(VehicleParam[2]), 'Height')
model.addParameter(Part_2_doc, "v", str(VehicleParam[3]), 'Volume')
model.addParameter(Part_2_doc, "o_f", str(VehicleParam[4]), 'Front Overhang')
model.addParameter(Part_2_doc, "o_r", str(VehicleParam[5]), 'Rear Overhang')
model.addParameter(Part_2_doc, "h_f", str(VehicleParam[6]), 'Front Wheel Well Height')
model.addParameter(Part_2_doc, "h_r", str(VehicleParam[7]), 'Rear Wheel Well Height')
model.addParameter(Part_2_doc, "r", str(VehicleParam[8]), 'Ride Height')
model.addParameter(Part_2_doc, "t", '(v/(w*h*0.85)+h)', 'Characteristic Trunk Size')
model.addParameter(Part_2_doc, "s", '-0.88665*t+0.68679', 'Trunk Shape Factor')
model.addParameter(Part_2_doc, "TunnelPlacement", str(TunnelParam[11]))


Point_66 = model.addPoint(Part_2_doc, "l/2", "(w*l)/2", "(h*l)")
Point_67 = model.addPoint(Part_2_doc, "-l/2", "0", "(r*l)")
Point_68 = model.addPoint(Part_2_doc, "-l*(1/2-(o_f)/3)", "0", "(r*l)")
Point_69 = model.addPoint(Part_2_doc, "-l/2", "0", "l*(r+h/8)")
Point_70 = model.addPoint(Part_2_doc, "-l/2", "0", "l*(r+h*11/20)")
Point_71 = model.addPoint(Part_2_doc, "-l/2+l/4*(h+1/2)*sin(80*pi/180)", "0",
    "l*(r+h*11/20)-cos(80*pi/180)")
Point_72 = model.addPoint(Part_2_doc, "-l/8", "0", "h*l")
Point_73 = model.addPoint(Part_2_doc, "0", "w*l/2", "h*l")
Point_74 = model.addPoint(Part_2_doc, "l*(1/2-o_r)", "0", "r*l")
Point_75 = model.addPoint(Part_2_doc, "l/2", "0", "r*l")
Point_76 = model.addPoint(Part_2_doc, "l/2", "0", "l*(r+h/10)")
Point_77 = model.addPoint(Part_2_doc, "l/2", "0", "l*(r+h/2)")
Point_78 = model.addPoint(Part_2_doc, "l/2", "0", "l*(r+h*5/8)")
Point_79 = model.addPoint(Part_2_doc, "l/2-(s*l)*sin((30+40*(1-s*h))*pi/180)", "0",
    "h*l-(s*l)*cos((30+40*(1-s*h))*pi/180)")
Point_80 = model.addPoint(Part_2_doc,
```

```
    "l/2-(s*l)*sin((30+40*(1-s*h))*pi/180)-((s*l)*cos((30+40*(1-s*h))*pi/180)*tan((30+15*(1-s/h))*pi/180))"
    "0", "h*l")
Point_81 = model.addPoint(Part_2_doc, "-l/2", "0", "h*l")
Point_82 = model.addPoint(Part_2_doc, "l/2", "0", "h*l")
Point_83 = model.addPoint(Part_2_doc, "0", "w*l/2", "1/3*h*l")
Point_84 = model.addPoint(Part_2_doc, "0", "w*l/2-w*l/5", "h*l")
Point_85 = model.addPoint(Part_2_doc, "-l/2", "w*l/2", "0")
Point_86 = model.addPoint(Part_2_doc, "l/2", "w*l/2", "0")
Point_87 = model.addPoint(Part_2_doc, "-l/2", "w*l/2-w*l/3", "0")
Point_88 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2", "0")
Point_89 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2", "0")
Point_90 = model.addPoint(Part_2_doc, "l/2", "0", "0")
Point_91 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2", "5/12*h_f*l")
Point_92 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2", "h_f*l")
Point_93 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2", "11/24*h_f*l")
Point_94 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2", "5/12*h_r*l")
Point_94.setName("Point_31")
Point_94.result().setName("Point_31")
Point_95 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2", "11/24*h_r*l")
Point_95.setName("Point_32")
Point_95.result().setName("Point_32")
Point_96 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2", "h_r*l")
Point_96.setName("Point_33")
Point_96.result().setName("Point_33")
Point_97 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2-l*0.012", "0")
Point_97.setName("Point_34")
Point_97.result().setName("Point_34")
Point_98 = model.addPoint(Part_2_doc, "-l/2+o_f*l", "w*l/2-l*0.048", "0")
Point_98.setName("Point_35")
Point_98.result().setName("Point_35")
Point_99 = model.addPoint(Part_2_doc, "-l/2+o_f*l-1/12*h_f*l", "w*l/2-l*0.015", "0")
Point_99.setName("Point_36")
Point_99.result().setName("Point_36")
Point_100 = model.addPoint(Part_2_doc, "-l/2+o_f*l+1/12*h_f*l", "w*l/2-l*0.015", "0")
Point_100.setName("Point_37")
Point_100.result().setName("Point_37")
Point_101 = model.addPoint(Part_2_doc, "-l/2+o_f*l-1/12*h_f*l", "w*l/2-l*0.045", "0")
Point_101.setName("Point_38")
Point_101.result().setName("Point_38")
Point_102 = model.addPoint(Part_2_doc, "-l/2+o_f*l+1/12*h_f*l", "w*l/2-0.045*l", "0")
Point_102.setName("Point_39")
Point_102.result().setName("Point_39")
Point_103 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2-l*0.012", "0")
Point_103.setName("Point_40")
Point_103.result().setName("Point_40")
```

```
Point_104 = model.addPoint(Part_2_doc, "l/2-o_r*l", "w*l/2-l*0.048", "0")
Point_104.setName("Point_41")
Point_104.result().setName("Point_41")
Point_105 = model.addPoint(Part_2_doc, "l/2-o_r*l+1/12*h_r*l", "w*l/2-l*0.015", "0")
Point_105.setName("Point_42")
Point_105.result().setName("Point_42")
Point_106 = model.addPoint(Part_2_doc, "l/2-o_r*l-1/12*h_r*l", "w*l/2-0.015*l", "0")
Point_106.setName("Point_43")
Point_106.result().setName("Point_43")
Point_107 = model.addPoint(Part_2_doc, "l/2-o_r*l+1/12*h_r*l", "w*l/2-0.045*l", "0")
Point_107.setName("Point_44")
Point_107.result().setName("Point_44")
Point_108 = model.addPoint(Part_2_doc, "l/2-o_r*l-1/12*h_f*l", "w/2*l-0.045*l", "0")
Point_108.setName("Point_45")
Point_108.result().setName("Point_45")
Point_109 = model.addPoint(Part_2_doc, "-l", "w*l", "-l*0.1")
Point_109.setName("Point_46")
Point_109.result().setName("Point_46")
Point_110 = model.addPoint(Part_2_doc, "3*l", "-w*l", "h*l*1.5")
Point_110.setName("Point_47")
Point_110.result().setName("Point_47")
Point_111 = model.addPoint(Part_2_doc, "-4*l", "-5*w*l", "0")
Point_111.setName("Point_48")
Point_111.result().setName("Point_48")
Point_112 = model.addPoint(Part_2_doc, "6*l", "5*w*l", "5*h*l")
Point_112.setName("Point_49")
Point_112.result().setName("Point_49")
Point_113 = model.addPoint(Part_2_doc, "-3*l", "-4*w*l", "0")
Point_113.setName("Point_50")
Point_113.result().setName("Point_50")
Point_114 = model.addPoint(Part_2_doc, "5*l", "4*w*l", "4*h*l")
Point_114.setName("Point_51")
Point_114.result().setName("Point_51")
Point_115 = model.addPoint(Part_2_doc, "-2*l", "-3*w*l", "0")
Point_115.setName("Point_52")
Point_115.result().setName("Point_52")
Point_116 = model.addPoint(Part_2_doc, "4*l", "3*w*l", "3*h*l")
Point_116.setName("Point_53")
Point_116.result().setName("Point_53")
Point_117 = model.addPoint(Part_2_doc, "-1.25*l/2", "-1.25*w*l/2", "0")
Point_117.setName("Point_54")
Point_117.result().setName("Point_54")
Point_118 = model.addPoint(Part_2_doc, "l*1.5", "1.25*w*l/2", "1.25*h*l")
Point_118.setName("Point_55")
Point_118.result().setName("Point_55")
```

```
Box_14 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_1"),
    model.selection("VERTEX", "Point_2"))
Box_15 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_46"),
    model.selection("VERTEX", "Point_47"))
Box_16 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_52"),
    model.selection("VERTEX", "Point_53"))
Box_17 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_50"),
    model.selection("VERTEX", "Point_51"))
Box_18 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_48"),
    model.selection("VERTEX", "Point_49"))
Box_19 = model.addBox(Part_2_doc, model.selection("VERTEX", "Point_54"),
    model.selection("VERTEX", "Point_55"))

Plane_7 = model.addPlane(Part_2_doc, model.selection("VERTEX", "Point_21"),
    model.selection("VERTEX", "Point_24"), model.selection("VERTEX", "Point_31"))

#
    ----------------------------------------------------------------------------------------------------
# t parameter determines the rear placement of the spoiler, with 0.62 being the approximate
    change between a sedan and a hatchback
# t>0.62 represents hatchback spoiler placement
# t<0.62 represents sedan spoiler placement
#
    ----------------------------------------------------------------------------------------------------

t = VehicleParam[3]/(VehicleParam[1]*VehicleParam[2]*0.85)+VehicleParam[2]
if t>= 0.62:
#
    ----------------------------------------------------------------------------------------------------
# HATCHBACK CREATION
#
    ----------------------------------------------------------------------------------------------------
    Sketch_11 = model.addSketch(Part_2_doc, model.defaultPlane("XOZ"))
    SketchBSpline_13 = Sketch_11.addSpline(poles=[(-1.763393333333333, 0.14007), (-2.03,
        0.14007), (-2.03, 0.32129825)])
    [SketchPoint_89, SketchPoint_90, SketchPoint_91] =
        SketchBSpline_13.controlPoles(auxiliary=[0, 1, 2])
    [SketchLine_116, SketchLine_117] = SketchBSpline_13.controlPolygon(auxiliary=[0, 1])
    SketchBSpline_14_poles = [(-2.03, 0.32129825),
                              (-2.03, 0.9374742999999999),
                              (-1.173260094016635, 0.7638261223330695),
                              (-0.5075, 1.449826),
                              (0, 1.449826)
                              ]
```

```
SketchBSpline_14 = Sketch_11.addSpline(poles=SketchBSpline_14_poles)
[SketchPoint_92, SketchPoint_93, SketchPoint_94, SketchPoint_95, SketchPoint_96] =
    SketchBSpline_14.controlPoles(
    auxiliary=[0, 1, 2, 3, 4])
[SketchLine_118, SketchLine_119, SketchLine_120, SketchLine_121] =
    SketchBSpline_14.controlPolygon(
    auxiliary=[0, 1, 2, 3])
SketchBSpline_15_poles = [(0, 1.449826),
                          (1.551607191560884, 1.449826),
                          (1.673039346430912, 1.310305442946679),
                          (2.03, 1.04621125),
                          (2.03, 0.8649829999999998)
                          ]
SketchBSpline_15 = Sketch_11.addSpline(poles=SketchBSpline_15_poles)
[SketchPoint_97, SketchPoint_98, SketchPoint_99, SketchPoint_100, SketchPoint_101] =
    SketchBSpline_15.controlPoles(
    auxiliary=[0, 1, 2, 3, 4])
[SketchLine_122, SketchLine_123, SketchLine_124, SketchLine_125] =
    SketchBSpline_15.controlPolygon(
    auxiliary=[0, 1, 2, 3])
SketchBSpline_16 = Sketch_11.addSpline(poles=[(2.03, 0.2850525999999999), (2.03,
    0.14007), (1.429932, 0.14007)],
                                       weights=[2, 1, 1])
[SketchPoint_102, SketchPoint_103, SketchPoint_104] =
    SketchBSpline_16.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_126, SketchLine_127] = SketchBSpline_16.controlPolygon(auxiliary=[0, 1])
SketchProjection_49 = Sketch_11.addProjection(model.selection("VERTEX", "Point_3"),
    False)
SketchPoint_105 = SketchProjection_49.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_89).coordinates(),
    SketchAPI_Point(SketchPoint_105).coordinates())
SketchProjection_50 = Sketch_11.addProjection(model.selection("VERTEX", "Point_2"),
    False)
SketchPoint_106 = SketchProjection_50.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
    SketchAPI_Point(SketchPoint_106).coordinates())
SketchProjection_51 = Sketch_11.addProjection(model.selection("VERTEX", "Point_4"),
    False)
SketchPoint_107 = SketchProjection_51.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_91).coordinates(),
    SketchAPI_Point(SketchPoint_107).coordinates())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_92).coordinates(),
    SketchAPI_Point(SketchPoint_91).coordinates())
SketchProjection_52 = Sketch_11.addProjection(model.selection("VERTEX", "Point_5"),
    False)
```

```
SketchPoint_108 = SketchProjection_52.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_118).endPoint(),
    SketchAPI_Point(SketchPoint_108).coordinates())
SketchProjection_53 = Sketch_11.addProjection(model.selection("VERTEX", "Point_6"),
    False)
SketchPoint_109 = SketchProjection_53.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_94).coordinates(),
    SketchAPI_Point(SketchPoint_109).coordinates())
SketchProjection_54 = Sketch_11.addProjection(model.selection("VERTEX", "Point_7"),
    False)
SketchPoint_110 = SketchProjection_54.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_95).coordinates(),
    SketchAPI_Point(SketchPoint_110).coordinates())
SketchProjection_55 = Sketch_11.addProjection(model.selection("VERTEX", "Point_8"),
    False)
SketchPoint_111 = SketchProjection_55.createdFeature()
Sketch_11.setCoincident(SketchBSpline_14.endPoint(),
    SketchAPI_Point(SketchPoint_111).coordinates())
Sketch_11.setCoincident(SketchBSpline_15.startPoint(), SketchBSpline_14.endPoint())
SketchProjection_56 = Sketch_11.addProjection(model.selection("VERTEX", "Point_15"),
    False)
SketchPoint_112 = SketchProjection_56.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_122).endPoint(),
    SketchAPI_Point(SketchPoint_112).coordinates())
SketchProjection_57 = Sketch_11.addProjection(model.selection("VERTEX", "Point_14"),
    False)
SketchPoint_113 = SketchProjection_57.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_99).coordinates(),
    SketchAPI_Point(SketchPoint_113).coordinates())
SketchProjection_58 = Sketch_11.addProjection(model.selection("VERTEX", "Point_13"),
    False)
SketchPoint_114 = SketchProjection_58.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_124).endPoint(),
    SketchAPI_Point(SketchPoint_114).coordinates())
SketchProjection_59 = Sketch_11.addProjection(model.selection("VERTEX", "Point_12"),
    False)
SketchPoint_115 = SketchProjection_59.createdFeature()
Sketch_11.setCoincident(SketchBSpline_15.endPoint(),
    SketchAPI_Point(SketchPoint_115).coordinates())
SketchProjection_60 = Sketch_11.addProjection(model.selection("VERTEX", "Point_11"),
    False)
SketchPoint_116 = SketchProjection_60.createdFeature()
Sketch_11.setCoincident(SketchBSpline_16.startPoint(),
    SketchAPI_Point(SketchPoint_116).coordinates())
SketchProjection_61 = Sketch_11.addProjection(model.selection("VERTEX", "Point_10"),
```

```
    False)
SketchPoint_117 = SketchProjection_61.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_127).startPoint(),
    SketchAPI_Point(SketchPoint_117).coordinates())
SketchProjection_62 = Sketch_11.addProjection(model.selection("VERTEX", "Point_9"),
    False)
SketchPoint_118 = SketchProjection_62.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_127).endPoint(),
    SketchAPI_Point(SketchPoint_118).coordinates())
SketchLine_128 = Sketch_11.addLine(2.03, 0.8649829999999998, 2.03, 0.2850525999999999)
Sketch_11.setCoincident(SketchBSpline_15.endPoint(), SketchLine_128.startPoint())
Sketch_11.setCoincident(SketchBSpline_16.startPoint(), SketchLine_128.endPoint())
SketchLine_129 = Sketch_11.addLine(-1.763393333333333, 0.14007, -2.03, 0.14007)
Sketch_11.setCoincident(SketchBSpline_13.startPoint(), SketchLine_129.startPoint())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
    SketchLine_129.endPoint())
SketchLine_130 = Sketch_11.addLine(-2.03, 0.14007, -2.03, 0.32129825)
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
    SketchLine_130.startPoint())
Sketch_11.setCoincident(SketchBSpline_13.endPoint(), SketchLine_130.endPoint())
SketchLine_131 = Sketch_11.addLine(-2.03, 0.32129825, -2.03, 1.449826)
Sketch_11.setCoincident(SketchBSpline_13.endPoint(), SketchLine_131.startPoint())
SketchProjection_63 = Sketch_11.addProjection(model.selection("VERTEX", "Point_16"),
    False)
SketchPoint_119 = SketchProjection_63.createdFeature()
Sketch_11.setCoincident(SketchLine_131.endPoint(), SketchPoint_119.result())
SketchLine_132 = Sketch_11.addLine(-2.03, 1.449826, 0, 1.449826)
Sketch_11.setCoincident(SketchLine_131.endPoint(), SketchLine_132.startPoint())
Sketch_11.setCoincident(SketchBSpline_14.endPoint(), SketchLine_132.endPoint())
SketchLine_133 = Sketch_11.addLine(0, 1.449826, 2.03, 1.449826)
Sketch_11.setCoincident(SketchBSpline_14.endPoint(), SketchLine_133.startPoint())
SketchProjection_64 = Sketch_11.addProjection(model.selection("VERTEX", "Point_17"),
    False)
SketchPoint_120 = SketchProjection_64.createdFeature()
Sketch_11.setCoincident(SketchLine_133.endPoint(), SketchPoint_120.result())
SketchLine_134 = Sketch_11.addLine(2.03, 1.449826, 2.03, 0.8649829999999998)
Sketch_11.setCoincident(SketchLine_133.endPoint(), SketchLine_134.startPoint())
Sketch_11.setCoincident(SketchBSpline_15.endPoint(), SketchLine_134.endPoint())
SketchLine_135 = Sketch_11.addLine(2.03, 0.2850525999999999, 2.03, 0.14007)
Sketch_11.setCoincident(SketchBSpline_16.startPoint(), SketchLine_135.startPoint())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_103).coordinates(),
    SketchLine_135.endPoint())
SketchLine_136 = Sketch_11.addLine(2.03, 0.14007, 1.429932, 0.14007)
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_103).coordinates(),
    SketchLine_136.startPoint())
```

```
Sketch_11.setCoincident(SketchBSpline_16.endPoint(), SketchLine_136.endPoint())
SketchLine_137 = Sketch_11.addLine(1.270102848723003, 1.449826, 1.574709453303134,
    1.368206906293568)
SketchLine_137.setName("SketchLine_32")
SketchLine_137.result().setName("SketchLine_32")
SketchLine_137.setAuxiliary(True)
SketchLine_138 = Sketch_11.addLine(1.574709453303134, 1.368206906293568,
    1.535492864755798, 1.37871495952473)
SketchLine_138.setName("SketchLine_33")
SketchLine_138.result().setName("SketchLine_33")
SketchLine_138.setAuxiliary(True)
Sketch_11.setCoincident(SketchLine_137.endPoint(), SketchLine_138.startPoint())
Sketch_11.setCoincident(SketchLine_138.endPoint(), SketchLine_137.result())
SketchLine_139 = Sketch_11.addLine(1.535492864755798, 1.37871495952473,
    1.697892864755798, 1.37871495952473)
SketchLine_139.setName("SketchLine_34")
SketchLine_139.result().setName("SketchLine_34")
SketchLine_139.setAuxiliary(True)
Sketch_11.setCoincident(SketchLine_138.endPoint(), SketchLine_139.startPoint())
Sketch_11.setHorizontal(SketchLine_139.result())
Sketch_11.setAngle(SketchLine_138.result(), SketchLine_139.result(), 15, type="Direct")
Sketch_11.setLength(SketchLine_138.result(), "l/100")
Sketch_11.setLength(SketchLine_139.result(), "l/25")
SketchLine_140 = Sketch_11.addLine(1.697892864755798, 1.37871495952473,
    0.9827608639438197, 0.1400700000000001)
SketchLine_140.setName("SketchLine_35")
SketchLine_140.result().setName("SketchLine_35")
Sketch_11.setCoincident(SketchLine_139.endPoint(), SketchLine_140.startPoint())
Sketch_11.setAngle(SketchLine_140.result(), SketchLine_139.result(), 60, type="Direct")
Sketch_11.setCoincident(SketchLine_137.startPoint(), SketchLine_133.result())
Sketch_11.setCoincident(SketchLine_138.endPoint(), SketchBSpline_15.result())
Sketch_11.setCoincident(SketchLine_137.endPoint(), SketchBSpline_15.result())
Sketch_11.setCoincident(SketchLine_140.endPoint(), SketchLine_136.result())

SketchBSpline_17_poles = [(0, spline),
                          (0.1, spline),
                          (0.2, spline),
                          (0.3, spline),
                          (0.4, spline)
                         ]
SketchBSpline_17 = Sketch_11.addSpline(poles=SketchBSpline_17_poles, weights=[1, 2, 2,
    2, 1])
SketchBSpline_17.setName("SketchBSpline_8")
SketchBSpline_17.result().setName("SketchBSpline_8")
[SketchPoint_121, SketchPoint_122, SketchPoint_123, SketchPoint_124, SketchPoint_125] =
```

```
    SketchBSpline_17.controlPoles(
    auxiliary=[0, 1, 2, 3, 4])
[SketchLine_141, SketchLine_142, SketchLine_143, SketchLine_144] =
    SketchBSpline_17.controlPolygon(
    auxiliary=[0, 1, 2, 3])
Sketch_11.setCoincident(SketchLine_132.endPoint(),
    SketchAPI_Point(SketchPoint_121).coordinates())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_122).coordinates(),
    SketchBSpline_15.result())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_123).coordinates(),
    SketchBSpline_15.result())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_124).coordinates(),
    SketchBSpline_15.result())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_125).coordinates(),
    SketchLine_140.startPoint())
Sketch_11.setEqual(SketchLine_141.result(), SketchLine_142.result())
Sketch_11.setEqual(SketchLine_142.result(), SketchLine_143.result())
Sketch_11.setEqual(SketchLine_143.result(), SketchLine_144.result())
model.do()
Sketch_11.changeFacesOrder([[SketchBSpline_13.result(), SketchLine_130.result(),
    SketchLine_129.result()],
                            [SketchBSpline_14.result(), SketchLine_132.result(),
                                SketchLine_131.result()],
                            [SketchBSpline_15.result(), SketchBSpline_17.result(),
                                SketchLine_140.result(),
                             SketchBSpline_15.result(), SketchLine_134.result(),
                                 SketchLine_133.result()],
                            [SketchBSpline_16.result(), SketchLine_136.result(),
                                SketchLine_135.result()],
                            [SketchBSpline_15.result(), SketchLine_140.result(),
                                SketchBSpline_17.result()],
                            [SketchBSpline_17.result(), SketchBSpline_15.result()]
                            ])
model.do()


Sketch_12 = model.addSketch(Part_2_doc, model.defaultPlane("YOZ"))
SketchBSpline_18 = Sketch_12.addSpline(
    poles=[(0.5160666, 1.449826), (0.860111, 1.449826), (0.860111,
        0.4832753333333332)], weights=[1, 1, 3])
SketchBSpline_18.setName("SketchBSpline_5")
SketchBSpline_18.result().setName("SketchBSpline_5")
[SketchPoint_126, SketchPoint_127, SketchPoint_128] =
    SketchBSpline_18.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_145, SketchLine_146] = SketchBSpline_18.controlPolygon(regular=[0, 1])
```

```
SketchProjection_65 = Sketch_12.addProjection(model.selection("VERTEX", "Point_19"),
    False)
SketchPoint_129 = SketchProjection_65.createdFeature()
SketchPoint_129.setName("SketchPoint_36")
SketchPoint_129.result().setName("SketchPoint_36")
Sketch_12.setCoincident(SketchAPI_Point(SketchPoint_126).coordinates(),
    SketchAPI_Point(SketchPoint_129).coordinates())
SketchProjection_66 = Sketch_12.addProjection(model.selection("VERTEX", "Point_8"),
    False)
SketchPoint_130 = SketchProjection_66.createdFeature()
SketchPoint_130.setName("SketchPoint_37")
SketchPoint_130.result().setName("SketchPoint_37")
Sketch_12.setCoincident(SketchAPI_Line(SketchLine_146).startPoint(),
    SketchAPI_Point(SketchPoint_130).coordinates())
SketchProjection_67 = Sketch_12.addProjection(model.selection("VERTEX", "Point_18"),
    False)
SketchPoint_131 = SketchProjection_67.createdFeature()
SketchPoint_131.setName("SketchPoint_38")
SketchPoint_131.result().setName("SketchPoint_38")
Sketch_12.setCoincident(SketchAPI_Point(SketchPoint_128).coordinates(),
    SketchAPI_Point(SketchPoint_131).coordinates())
model.do()


Sketch_13 = model.addSketch(Part_2_doc, model.defaultPlane("XOY"))
SketchBSpline_19 = Sketch_13.addSpline(poles=[(-1.23018, 0.860111), (-2.03, 0.860111),
    (-2.03, 0.2867036666666667)])
SketchBSpline_19.setName("SketchBSpline_6")
SketchBSpline_19.result().setName("SketchBSpline_6")
[SketchPoint_132, SketchPoint_133, SketchPoint_134] =
    SketchBSpline_19.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_147, SketchLine_148] = SketchBSpline_19.controlPolygon(regular=[0, 1])
SketchBSpline_20 = Sketch_13.addSpline(poles=[(1.429932, 0.860111), (2.03, 0.860111),
    (2.03, 0)], weights=[1, 3, 1])
SketchBSpline_20.setName("SketchBSpline_7")
SketchBSpline_20.result().setName("SketchBSpline_7")
[SketchPoint_135, SketchPoint_136, SketchPoint_137] =
    SketchBSpline_20.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_149, SketchLine_150] = SketchBSpline_20.controlPolygon(regular=[0, 1])
SketchProjection_68 = Sketch_13.addProjection(model.selection("VERTEX", "Point_22"),
    False)
SketchPoint_138 = SketchProjection_68.createdFeature()
SketchPoint_138.setName("SketchPoint_45")
SketchPoint_138.result().setName("SketchPoint_45")
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_134).coordinates(),
```

```
    SketchAPI_Point(SketchPoint_138).coordinates())
SketchProjection_69 = Sketch_13.addProjection(model.selection("VERTEX", "Point_20"),
    False)
SketchPoint_139 = SketchProjection_69.createdFeature()
SketchPoint_139.setName("SketchPoint_46")
SketchPoint_139.result().setName("SketchPoint_46")
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_133).coordinates(),
    SketchAPI_Point(SketchPoint_139).coordinates())
SketchProjection_70 = Sketch_13.addProjection(model.selection("VERTEX", "Point_23"),
    False)
SketchPoint_140 = SketchProjection_70.createdFeature()
SketchPoint_140.setName("SketchPoint_47")
SketchPoint_140.result().setName("SketchPoint_47")
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_132).coordinates(),
    SketchAPI_Point(SketchPoint_140).coordinates())
SketchProjection_71 = Sketch_13.addProjection(model.selection("VERTEX", "Point_24"),
    False)
SketchPoint_141 = SketchProjection_71.createdFeature()
SketchPoint_141.setName("SketchPoint_48")
SketchPoint_141.result().setName("SketchPoint_48")
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_135).coordinates(),
    SketchAPI_Point(SketchPoint_141).coordinates())
SketchProjection_72 = Sketch_13.addProjection(model.selection("VERTEX", "Point_21"),
    False)
SketchPoint_142 = SketchProjection_72.createdFeature()
SketchPoint_142.setName("SketchPoint_49")
SketchPoint_142.result().setName("SketchPoint_49")
Sketch_13.setCoincident(SketchAPI_Line(SketchLine_149).endPoint(),
    SketchAPI_Point(SketchPoint_142).coordinates())
SketchProjection_73 = Sketch_13.addProjection(model.selection("VERTEX", "Point_25"),
    False)
SketchPoint_143 = SketchProjection_73.createdFeature()
SketchPoint_143.setName("SketchPoint_50")
SketchPoint_143.result().setName("SketchPoint_50")
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_137).coordinates(),
    SketchAPI_Point(SketchPoint_143).coordinates())
model.do()


Sketch_14 = model.addSketch(Part_2_doc, model.selection("FACE", "Plane_1"))
SketchProjection_74 = Sketch_14.addProjection(model.selection("VERTEX", "Point_32"),
    False)
SketchPoint_144 = SketchProjection_74.createdFeature()
SketchPoint_144.setName("SketchPoint_51")
SketchPoint_144.result().setName("SketchPoint_51")
```

```
SketchProjection_75 = Sketch_14.addProjection(model.selection("VERTEX", "Point_24"),
    False)
SketchPoint_145 = SketchProjection_75.createdFeature()
SketchPoint_145.setName("SketchPoint_52")
SketchPoint_145.result().setName("SketchPoint_52")
SketchCircle_1 = Sketch_14.addCircle(1.429932, -0.2982915833333333, 0.2982915833333333)
Sketch_14.setCoincident(SketchPoint_144.result(), SketchCircle_1.center())
Sketch_14.setCoincident(SketchPoint_145.result(), SketchCircle_1.results()[1])
SketchProjection_76 = Sketch_14.addProjection(model.selection("VERTEX", "Point_31"),
    False)
SketchPoint_146 = SketchProjection_76.createdFeature()
SketchPoint_146.setName("SketchPoint_53")
SketchPoint_146.result().setName("SketchPoint_53")
SketchProjection_77 = Sketch_14.addProjection(model.selection("VERTEX", "Point_33"),
    False)
SketchPoint_147 = SketchProjection_77.createdFeature()
SketchPoint_147.setName("SketchPoint_54")
SketchPoint_147.result().setName("SketchPoint_54")
SketchEllipse_1 = Sketch_14.addEllipse(1.429932, -0.2711741666666667, 1.429932,
    -0.3976704483433147, 0.3579498999999999)
[SketchPoint_148, SketchPoint_149, SketchPoint_150, SketchPoint_151, SketchPoint_152,
    SketchPoint_153, SketchPoint_154,
 SketchLine_151, SketchLine_152] = SketchEllipse_1.construction(center="aux",
     firstFocus="aux", secondFocus="aux",
                                                                majorAxisStart="aux",
                                                                    majorAxisEnd="aux",
                                                                minorAxisStart="aux",
                                                                    minorAxisEnd="aux",
                                                                majorAxis="aux",
                                                                    minorAxis="aux")
Sketch_14.setCoincident(SketchPoint_146.result(), SketchEllipse_1.center())
Sketch_14.setCoincident(SketchPoint_147.result(), SketchEllipse_1.majorAxisPositive())
Sketch_14.setHorizontalDistance(SketchAPI_Point(SketchPoint_146).coordinates(),
                                SketchAPI_Point(SketchPoint_153).coordinates(),
                                    "1.1*h_r*l/2")
SketchProjection_78 = Sketch_14.addProjection(model.selection("VERTEX", "Point_28"),
    False)
SketchPoint_155 = SketchProjection_78.createdFeature()
SketchPoint_155.setName("SketchPoint_62")
SketchPoint_155.result().setName("SketchPoint_62")
SketchProjection_79 = Sketch_14.addProjection(model.selection("VERTEX", "Point_23"),
    False)
SketchPoint_156 = SketchProjection_79.createdFeature()
SketchPoint_156.setName("SketchPoint_63")
SketchPoint_156.result().setName("SketchPoint_63")
```

```
SketchCircle_2 = Sketch_14.addCircle(-1.23018, -0.2956864166666666, 0.2956864166666666)
Sketch_14.setCoincident(SketchPoint_155.result(), SketchCircle_2.center())
Sketch_14.setCoincident(SketchPoint_156.result(), SketchCircle_2.results()[1])
SketchProjection_80 = Sketch_14.addProjection(model.selection("VERTEX", "Point_26"),
    False)
SketchPoint_157 = SketchProjection_80.createdFeature()
SketchPoint_157.setName("SketchPoint_64")
SketchPoint_157.result().setName("SketchPoint_64")
SketchProjection_81 = Sketch_14.addProjection(model.selection("VERTEX", "Point_27"),
    False)
SketchPoint_158 = SketchProjection_81.createdFeature()
SketchPoint_158.setName("SketchPoint_65")
SketchPoint_158.result().setName("SketchPoint_65")
SketchEllipse_2 = Sketch_14.addEllipse(-1.23018, -0.2688058333333334, -1.23018,
    -0.3941973439909709, 0.3548237)
[SketchPoint_159, SketchPoint_160, SketchPoint_161, SketchPoint_162, SketchPoint_163,
    SketchPoint_164, SketchPoint_165,
 SketchLine_153, SketchLine_154] = SketchEllipse_2.construction(center="aux",
     firstFocus="aux", secondFocus="aux",
                                                                majorAxisStart="aux",
                                                                    majorAxisEnd="aux",
                                                                minorAxisStart="aux",
                                                                    minorAxisEnd="aux",
                                                                majorAxis="aux",
                                                                    minorAxis="aux")
Sketch_14.setCoincident(SketchPoint_157.result(), SketchEllipse_2.center())
Sketch_14.setCoincident(SketchPoint_158.result(), SketchEllipse_2.majorAxisPositive())
Sketch_14.setHorizontalDistance(SketchAPI_Point(SketchPoint_157).coordinates(),
                                SketchAPI_Line(SketchLine_154).endPoint(),
                                    "1.1*h_f*l/2")
model.do()


ExtrusionCut_7_objects_1 = [model.selection("COMPOUND", "Sketch_2"),
                            model.selection("COMPOUND", "Sketch_3"),
                            model.selection("FACE",
                                            "Sketch_1/Face-SketchBSpline_3f-SketchBSpline_8f-SketchLine
                            model.selection("FACE",
                                "Sketch_1/Face-SketchBSpline_4f-SketchLine_21r-SketchLine_20r"),
                            model.selection("FACE",
                                "Sketch_1/Face-SketchBSpline_1r-SketchLine_15r-SketchLine_14r"),
                            model.selection("FACE",
                                "Sketch_1/Face-SketchBSpline_2f-SketchLine_17r-SketchLine_16r")]
ExtrusionCut_7 = model.addExtrusionCut(Part_2_doc, ExtrusionCut_7_objects_1,
    model.selection(),
```

```
                                        [model.selection("SOLID", "Box_1_1")])
ExtrusionCut_8 = model.addExtrusionCut(Part_2_doc, [model.selection("COMPOUND",
    "Sketch_4")], model.selection(), 0,
                                        "l*0.075", [model.selection("SOLID",
                                            "ExtrusionCut_1_1")])
Extrusion_1 = model.addExtrusion(Part_2_doc, [model.selection("FACE",
    "Sketch_4/Face-SketchCircle_1_2f")],
                                    model.selection(), "-l*0.005", "l*0.055",
                                        "Faces|Wires")
Extrusion_1.setName("Extrusion_3")
Extrusion_1.result().setName("Extrusion_3_1")
Extrusion_2 = model.addExtrusion(Part_2_doc, [model.selection("FACE",
    "Sketch_4/Face-SketchCircle_2_2f")],
                                    model.selection(), "-l*0.005", "l*0.055",
                                        "Faces|Wires")
Extrusion_2.setName("Extrusion_4")
Extrusion_2.result().setName("Extrusion_4_1")
Fillet_3 = model.addFillet(Part_2_doc, [
    model.selection("EDGE",
        "[Extrusion_3_1/Generated_Face&Sketch_4/SketchCircle_1_2][Extrusion_3_1/To_Face]"),
    model.selection("FACE", "Extrusion_3_1/Generated_Face&Sketch_4/SketchCircle_1_2")],
        "0.01*l", keepSubResults=True)
Fillet_4 = model.addFillet(Part_2_doc, [
    model.selection("EDGE",
        "[Extrusion_4_1/Generated_Face&Sketch_4/SketchCircle_2_2][Extrusion_4_1/From_Face]"),
    model.selection("EDGE",
        "[Extrusion_4_1/Generated_Face&Sketch_4/SketchCircle_2_2][Extrusion_4_1/To_Face]")],
                            "0.01*l", keepSubResults=True)
Fillet_5 = model.addFillet(Part_2_doc, [model.selection("EDGE",
                                                        "[ExtrusionCut_2_1/Modified_Face&Sketch_2/Sket
                                        model.selection("EDGE",
                                                        "[ExtrusionCut_2_1/Modified_Face&Sketch_3/Sket
                            bigFilletRadius, keepSubResults=True)
Fillet_6_objects = [model.selection("EDGE",
                                    "[Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5][Fillet_3_1/MF:Fil
                    model.selection("EDGE",
                                    "[Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5][ExtrusionCut_1_1/
                    model.selection("EDGE",
                                    "[ExtrusionCut_1_1/Generated_Face&Sketch_1/SketchBSpline_8][Extrus
                    model.selection("EDGE",
                                    "[Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5][ExtrusionCut_1_1/
                    model.selection("EDGE",
                                    "[ExtrusionCut_2_1/Modified_Face&Sketch_1/SketchBSpline_4][Fillet_3
                    model.selection("FACE",
                        "Fillet_3_1/MF:Fillet&Sketch_4/SketchEllipse_1"),
```

```
                        model.selection("FACE",
                                        "(Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5)(ExtrusionCut_2_1/
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_2"),
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_1"),
                        model.selection("FACE",
                            "Fillet_3_1/MF:Fillet&Sketch_4/SketchEllipse_2"),
                        model.selection("FACE",
                                        "(Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5)(Fillet_3_1/MF:Fil
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_3"),
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_4"),
                        model.selection("FACE",
                            "Fillet_3_1/MF:Fillet&Sketch_3/SketchBSpline_6"),
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&Box_1_1/Right")]
    Fillet_6 = model.addFillet(Part_2_doc, Fillet_6_objects, filletRadius,
        keepSubResults=True)
    Symmetry_1_objects = [model.selection("SOLID", "Fillet_1_1"),
                          model.selection("SOLID", "Fillet_2_1"),
                          model.selection("SOLID", "Fillet_4_1")]
    Symmetry_1 = model.addSymmetry(Part_2_doc, Symmetry_1_objects, model.selection("FACE",
        "PartSet/XOZ"),
                                   keepOriginal=True, keepSubResults=True)
    Fuse_2 = model.addFuse(Part_2_doc,
                           [model.selection("SOLID", "Symmetry_1_3_1"),
                               model.selection("SOLID", "Symmetry_1_3_2")],
                           removeEdges=True, keepSubResults=True)
else:
    #
        ----------------------------------------------------------------------------------------------
    # SEDAN CREATION
    #
        ----------------------------------------------------------------------------------------------
    Sketch_11 = model.addSketch(Part_2_doc, model.defaultPlane("XOZ"))
    SketchBSpline_13 = Sketch_11.addSpline(
        poles=[(-2.1043528, 0.1470084), (-2.442, 0.1470084), (-2.442, 0.3333940500000001)])
    [SketchPoint_89, SketchPoint_90, SketchPoint_91] =
        SketchBSpline_13.controlPoles(auxiliary=[0, 1, 2])
    [SketchLine_116, SketchLine_117] = SketchBSpline_13.controlPolygon(auxiliary=[0, 1])
    SketchBSpline_14_poles = [(-2.442, 0.3333940500000001),
                              (-2.442, 0.9671052600000001),
                              (-1.473666800445607, 0.7934570823330697),
```

```
                          (-0.6105, 1.4910852),
                          (0, 1.4910852)
                          ]
SketchBSpline_14 = Sketch_11.addSpline(poles=SketchBSpline_14_poles)
[SketchPoint_92, SketchPoint_93, SketchPoint_94, SketchPoint_95, SketchPoint_96] =
    SketchBSpline_14.controlPoles(
    auxiliary=[0, 1, 2, 3, 4])
[SketchLine_118, SketchLine_119, SketchLine_120, SketchLine_121] =
    SketchBSpline_14.controlPolygon(
    auxiliary=[0, 1, 2, 3])
SketchBSpline_15_poles = [(0, 1.4910852),
                          (1.485286957193869, 1.4910852),
                          (1.708663283373133, 1.195135389512772),
                          (2.442, 1.07893665),
                          (2.442, 0.8925510000000001)
                          ]
SketchBSpline_15 = Sketch_11.addSpline(poles=SketchBSpline_15_poles)
[SketchPoint_97, SketchPoint_98, SketchPoint_99, SketchPoint_100, SketchPoint_101] =
    SketchBSpline_15.controlPoles(
    auxiliary=[0, 1, 2, 3, 4])
[SketchLine_122, SketchLine_123, SketchLine_124, SketchLine_125] =
    SketchBSpline_15.controlPolygon(
    auxiliary=[0, 1, 2, 3])
SketchBSpline_16 = Sketch_11.addSpline(
    poles=[(2.442, 0.2961169200000001), (2.442, 0.1470084), (1.3518912, 0.1470084)],
        weights=[2, 1, 1])
[SketchPoint_102, SketchPoint_103, SketchPoint_104] =
    SketchBSpline_16.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_126, SketchLine_127] = SketchBSpline_16.controlPolygon(auxiliary=[0, 1])
SketchProjection_49 = Sketch_11.addProjection(model.selection("VERTEX", "Point_3"),
    False)
SketchPoint_105 = SketchProjection_49.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_89).coordinates(),
                        SketchAPI_Point(SketchPoint_105).coordinates())
SketchProjection_50 = Sketch_11.addProjection(model.selection("VERTEX", "Point_2"),
    False)
SketchPoint_106 = SketchProjection_50.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
                        SketchAPI_Point(SketchPoint_106).coordinates())
SketchProjection_51 = Sketch_11.addProjection(model.selection("VERTEX", "Point_4"),
    False)
SketchPoint_107 = SketchProjection_51.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_91).coordinates(),
                        SketchAPI_Point(SketchPoint_107).coordinates())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_92).coordinates(),
```

```
                        SketchAPI_Point(SketchPoint_91).coordinates())
SketchProjection_52 = Sketch_11.addProjection(model.selection("VERTEX", "Point_5"),
    False)
SketchPoint_108 = SketchProjection_52.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_118).endPoint(),
    SketchAPI_Point(SketchPoint_108).coordinates())
SketchProjection_53 = Sketch_11.addProjection(model.selection("VERTEX", "Point_6"),
    False)
SketchPoint_109 = SketchProjection_53.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_94).coordinates(),
                        SketchAPI_Point(SketchPoint_109).coordinates())
SketchProjection_54 = Sketch_11.addProjection(model.selection("VERTEX", "Point_7"),
    False)
SketchPoint_110 = SketchProjection_54.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_95).coordinates(),
                        SketchAPI_Point(SketchPoint_110).coordinates())
SketchProjection_55 = Sketch_11.addProjection(model.selection("VERTEX", "Point_8"),
    False)
SketchPoint_111 = SketchProjection_55.createdFeature()
Sketch_11.setCoincident(SketchBSpline_14.endPoint(),
    SketchAPI_Point(SketchPoint_111).coordinates())
Sketch_11.setCoincident(SketchBSpline_15.startPoint(), SketchBSpline_14.endPoint())
SketchProjection_56 = Sketch_11.addProjection(model.selection("VERTEX", "Point_15"),
    False)
SketchPoint_112 = SketchProjection_56.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_122).endPoint(),
    SketchAPI_Point(SketchPoint_112).coordinates())
SketchProjection_57 = Sketch_11.addProjection(model.selection("VERTEX", "Point_14"),
    False)
SketchPoint_113 = SketchProjection_57.createdFeature()
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_99).coordinates(),
                        SketchAPI_Point(SketchPoint_113).coordinates())
SketchProjection_58 = Sketch_11.addProjection(model.selection("VERTEX", "Point_13"),
    False)
SketchPoint_114 = SketchProjection_58.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_124).endPoint(),
    SketchAPI_Point(SketchPoint_114).coordinates())
SketchProjection_59 = Sketch_11.addProjection(model.selection("VERTEX", "Point_12"),
    False)
SketchPoint_115 = SketchProjection_59.createdFeature()
Sketch_11.setCoincident(SketchBSpline_15.endPoint(),
    SketchAPI_Point(SketchPoint_115).coordinates())
SketchProjection_60 = Sketch_11.addProjection(model.selection("VERTEX", "Point_11"),
    False)
SketchPoint_116 = SketchProjection_60.createdFeature()
```

```
Sketch_11.setCoincident(SketchBSpline_16.startPoint(),
    SketchAPI_Point(SketchPoint_116).coordinates())
SketchProjection_61 = Sketch_11.addProjection(model.selection("VERTEX", "Point_10"),
    False)
SketchPoint_117 = SketchProjection_61.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_127).startPoint(),
    SketchAPI_Point(SketchPoint_117).coordinates())
SketchProjection_62 = Sketch_11.addProjection(model.selection("VERTEX", "Point_9"),
    False)
SketchPoint_118 = SketchProjection_62.createdFeature()
Sketch_11.setCoincident(SketchAPI_Line(SketchLine_127).endPoint(),
    SketchAPI_Point(SketchPoint_118).coordinates())
SketchLine_128 = Sketch_11.addLine(2.442, 0.8925510000000001, 2.442, 0.2961169200000001)
Sketch_11.setCoincident(SketchBSpline_15.endPoint(), SketchLine_128.startPoint())
Sketch_11.setCoincident(SketchBSpline_16.startPoint(), SketchLine_128.endPoint())
SketchLine_129 = Sketch_11.addLine(-2.1043528, 0.1470084, -2.442, 0.1470084)
Sketch_11.setCoincident(SketchBSpline_13.startPoint(), SketchLine_129.startPoint())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
    SketchLine_129.endPoint())
SketchLine_130 = Sketch_11.addLine(-2.442, 0.1470084, -2.442, 0.3333940500000001)
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_90).coordinates(),
    SketchLine_130.startPoint())
Sketch_11.setCoincident(SketchBSpline_13.endPoint(), SketchLine_130.endPoint())
SketchLine_131 = Sketch_11.addLine(-2.442, 0.3333940500000001, -2.442, 1.4910852)
Sketch_11.setCoincident(SketchBSpline_13.endPoint(), SketchLine_131.startPoint())
SketchProjection_63 = Sketch_11.addProjection(model.selection("VERTEX", "Point_16"),
    False)
SketchPoint_119 = SketchProjection_63.createdFeature()
Sketch_11.setCoincident(SketchLine_131.endPoint(), SketchPoint_119.result())
SketchLine_132 = Sketch_11.addLine(-2.442, 1.4910852, 0, 1.4910852)
Sketch_11.setCoincident(SketchLine_131.endPoint(), SketchLine_132.startPoint())
Sketch_11.setCoincident(SketchBSpline_14.endPoint(), SketchLine_132.endPoint())
SketchLine_133 = Sketch_11.addLine(0, 1.4910852, 2.442, 1.4910852)
Sketch_11.setCoincident(SketchBSpline_14.endPoint(), SketchLine_133.startPoint())
SketchProjection_64 = Sketch_11.addProjection(model.selection("VERTEX", "Point_17"),
    False)
SketchPoint_120 = SketchProjection_64.createdFeature()
Sketch_11.setCoincident(SketchLine_133.endPoint(), SketchPoint_120.result())
SketchLine_134 = Sketch_11.addLine(2.442, 1.4910852, 2.442, 0.8925510000000001)
Sketch_11.setCoincident(SketchLine_133.endPoint(), SketchLine_134.startPoint())
Sketch_11.setCoincident(SketchBSpline_15.endPoint(), SketchLine_134.endPoint())
SketchLine_135 = Sketch_11.addLine(2.442, 0.2961169200000001, 2.442, 0.1470084)
Sketch_11.setCoincident(SketchBSpline_16.startPoint(), SketchLine_135.startPoint())
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_103).coordinates(),
    SketchLine_135.endPoint())
```

```
SketchLine_136 = Sketch_11.addLine(2.442, 0.1470084, 1.3518912, 0.1470084)
Sketch_11.setCoincident(SketchAPI_Point(SketchPoint_103).coordinates(),
    SketchLine_136.startPoint())
Sketch_11.setCoincident(SketchBSpline_16.endPoint(), SketchLine_136.endPoint())
SketchLine_137 = Sketch_11.addLine(2.24664, 1.089938918364612, 2.442, 1.089938918364612)
SketchLine_137.setName("SketchLine_40")
SketchLine_137.result().setName("SketchLine_40")
Sketch_11.setCoincident(SketchLine_137.startPoint(), SketchBSpline_15.result())
Sketch_11.setCoincident(SketchLine_137.endPoint(), SketchLine_134.result())
Sketch_11.setHorizontal(SketchLine_137.result())
Sketch_11.setLength(SketchLine_137.result(), "l/25")
model.do()
Sketch_11.changeFacesOrder([[SketchBSpline_13.result(), SketchLine_130.result(),
    SketchLine_129.result()],
                            [SketchBSpline_14.result(), SketchLine_132.result(),
                                SketchLine_131.result()],
                            [SketchBSpline_15.result(), SketchLine_137.result(),
                                SketchLine_134.result(),
                             SketchLine_133.result()],
                            [SketchBSpline_16.result(), SketchLine_136.result(),
                                SketchLine_135.result()],
                            [SketchBSpline_15.result(), SketchLine_134.result(),
                                SketchLine_137.result()]
                            ])
model.do()


Sketch_12 = model.addSketch(Part_2_doc, model.defaultPlane("YOZ"))
SketchBSpline_17 = Sketch_12.addSpline(
    poles=[(0.56131812, spline), (0.9355302000000001, spline), (0.9355302000000001,
        spline)],
    weights=[1, 1, 3])
[SketchPoint_121, SketchPoint_122, SketchPoint_123] =
    SketchBSpline_17.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_138, SketchLine_139] = SketchBSpline_17.controlPolygon(regular=[0, 1])
SketchProjection_65 = Sketch_12.addProjection(model.selection("VERTEX", "Point_19"),
    False)
SketchPoint_124 = SketchProjection_65.createdFeature()
Sketch_12.setCoincident(SketchAPI_Point(SketchPoint_121).coordinates(),
                        SketchAPI_Point(SketchPoint_124).coordinates())
SketchProjection_66 = Sketch_12.addProjection(model.selection("VERTEX", "Point_8"),
    False)
SketchPoint_125 = SketchProjection_66.createdFeature()
Sketch_12.setCoincident(SketchAPI_Line(SketchLine_139).startPoint(),
    SketchAPI_Point(SketchPoint_125).coordinates())
```

```
SketchProjection_67 = Sketch_12.addProjection(model.selection("VERTEX", "Point_18"),
    False)
SketchPoint_126 = SketchProjection_67.createdFeature()
Sketch_12.setCoincident(SketchAPI_Point(SketchPoint_123).coordinates(),
                        SketchAPI_Point(SketchPoint_126).coordinates())
model.do()


Sketch_13 = model.addSketch(Part_2_doc, model.defaultPlane("XOY"))
SketchBSpline_18 = Sketch_13.addSpline(
    poles=[(-1.4290584, 0.9355302000000001), (-2.442, 0.9355302000000001), (-2.442,
        0.3118434)])
[SketchPoint_127, SketchPoint_128, SketchPoint_129] =
    SketchBSpline_18.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_140, SketchLine_141] = SketchBSpline_18.controlPolygon(regular=[0, 1])
SketchBSpline_19 = Sketch_13.addSpline(
    poles=[(1.3518912, 0.9355302000000001), (2.442, 0.9355302000000001), (2.442, 0)],
        weights=[1, 3, 1])
[SketchPoint_130, SketchPoint_131, SketchPoint_132] =
    SketchBSpline_19.controlPoles(auxiliary=[0, 1, 2])
[SketchLine_142, SketchLine_143] = SketchBSpline_19.controlPolygon(regular=[0, 1])
SketchProjection_68 = Sketch_13.addProjection(model.selection("VERTEX", "Point_22"),
    False)
SketchPoint_133 = SketchProjection_68.createdFeature()
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_129).coordinates(),
                        SketchAPI_Point(SketchPoint_133).coordinates())
SketchProjection_69 = Sketch_13.addProjection(model.selection("VERTEX", "Point_20"),
    False)
SketchPoint_134 = SketchProjection_69.createdFeature()
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_128).coordinates(),
                        SketchAPI_Point(SketchPoint_134).coordinates())
SketchProjection_70 = Sketch_13.addProjection(model.selection("VERTEX", "Point_23"),
    False)
SketchPoint_135 = SketchProjection_70.createdFeature()
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_127).coordinates(),
                        SketchAPI_Point(SketchPoint_135).coordinates())
SketchProjection_71 = Sketch_13.addProjection(model.selection("VERTEX", "Point_24"),
    False)
SketchPoint_136 = SketchProjection_71.createdFeature()
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_130).coordinates(),
                        SketchAPI_Point(SketchPoint_136).coordinates())
SketchProjection_72 = Sketch_13.addProjection(model.selection("VERTEX", "Point_21"),
    False)
SketchPoint_137 = SketchProjection_72.createdFeature()
Sketch_13.setCoincident(SketchAPI_Line(SketchLine_142).endPoint(),
```

```
    SketchAPI_Point(SketchPoint_137).coordinates())
SketchProjection_73 = Sketch_13.addProjection(model.selection("VERTEX", "Point_25"),
    False)
SketchPoint_138 = SketchProjection_73.createdFeature()
Sketch_13.setCoincident(SketchAPI_Point(SketchPoint_132).coordinates(),
                        SketchAPI_Point(SketchPoint_138).coordinates())
model.do()


Sketch_14 = model.addSketch(Part_2_doc, model.selection("FACE", "Plane_1"))
SketchProjection_74 = Sketch_14.addProjection(model.selection("VERTEX", "Point_32"),
    False)
SketchPoint_139 = SketchProjection_74.createdFeature()
SketchProjection_75 = Sketch_14.addProjection(model.selection("VERTEX", "Point_24"),
    False)
SketchPoint_140 = SketchProjection_75.createdFeature()
SketchCircle_1 = Sketch_14.addCircle(1.3518912, -0.32077705, 0.32077705)
Sketch_14.setCoincident(SketchPoint_139.result(), SketchCircle_1.center())
Sketch_14.setCoincident(SketchPoint_140.result(), SketchCircle_1.results()[1])
SketchProjection_76 = Sketch_14.addProjection(model.selection("VERTEX", "Point_31"),
    False)
SketchPoint_141 = SketchProjection_76.createdFeature()
SketchProjection_77 = Sketch_14.addProjection(model.selection("VERTEX", "Point_33"),
    False)
SketchPoint_142 = SketchProjection_77.createdFeature()
SketchEllipse_1 = Sketch_14.addEllipse(1.3518912, -0.2916155000000001,
    1.351891200027085, -0.4276471762027652,
                                       0.3849324599700071)
[SketchPoint_143, SketchPoint_144, SketchPoint_145, SketchPoint_146, SketchPoint_147,
    SketchPoint_148,
 SketchPoint_149, SketchLine_144, SketchLine_145] =
     SketchEllipse_1.construction(center="aux", firstFocus="aux",
                                                                    secondFocus="aux",
                                                                    majorAxisStart="aux",
                                                                    majorAxisEnd="aux",
                                                                    minorAxisStart="aux",
                                                                    minorAxisEnd="aux",
                                                                    majorAxis="aux",
                                                                        minorAxis="aux")
Sketch_14.setCoincident(SketchPoint_141.result(), SketchEllipse_1.center())
Sketch_14.setCoincident(SketchPoint_142.result(), SketchEllipse_1.majorAxisPositive())
Sketch_14.setHorizontalDistance(SketchAPI_Point(SketchPoint_141).coordinates(),
                                SketchAPI_Point(SketchPoint_148).coordinates(),
                                    "1.1*h_r*l/2")
SketchProjection_78 = Sketch_14.addProjection(model.selection("VERTEX", "Point_28"),
```

```
    False)
SketchPoint_150 = SketchProjection_78.createdFeature()
SketchProjection_79 = Sketch_14.addProjection(model.selection("VERTEX", "Point_23"),
    False)
SketchPoint_151 = SketchProjection_79.createdFeature()
SketchCircle_2 = Sketch_14.addCircle(-1.4290584, -0.31898625, 0.31898625)
Sketch_14.setCoincident(SketchPoint_150.result(), SketchCircle_2.center())
Sketch_14.setCoincident(SketchPoint_151.result(), SketchCircle_2.results()[1])
SketchProjection_80 = Sketch_14.addProjection(model.selection("VERTEX", "Point_26"),
    False)
SketchPoint_152 = SketchProjection_80.createdFeature()
SketchProjection_81 = Sketch_14.addProjection(model.selection("VERTEX", "Point_27"),
    False)
SketchPoint_153 = SketchProjection_81.createdFeature()
SketchEllipse_2 = Sketch_14.addEllipse(-1.4290584, -0.2899875, -1.429058400000009,
    -0.4252597530085214,
                                       0.3827834999999899)
[SketchPoint_154, SketchPoint_155, SketchPoint_156, SketchPoint_157, SketchPoint_158,
    SketchPoint_159,
 SketchPoint_160, SketchLine_146, SketchLine_147] =
     SketchEllipse_2.construction(center="aux", firstFocus="aux",
                                                                    secondFocus="aux",
                                                                    majorAxisStart="aux",
                                                                    majorAxisEnd="aux",
                                                                    minorAxisStart="aux",
                                                                    minorAxisEnd="aux",
                                                                    majorAxis="aux",
                                                                        minorAxis="aux")
Sketch_14.setCoincident(SketchPoint_152.result(), SketchEllipse_2.center())
Sketch_14.setCoincident(SketchPoint_153.result(), SketchEllipse_2.majorAxisPositive())
Sketch_14.setHorizontalDistance(SketchAPI_Point(SketchPoint_152).coordinates(),
                                SketchAPI_Line(SketchLine_147).endPoint(),
                                    "1.1*h_f*l/2")
model.do()


ExtrusionCut_7_objects_1 = [model.selection("COMPOUND", "Sketch_2"),
                            model.selection("COMPOUND", "Sketch_3"),
                            model.selection("FACE",
                                            "Sketch_1/Face-SketchBSpline_3f-SketchLine_40f-SketchLine_1
                            model.selection("FACE",
                                "Sketch_1/Face-SketchBSpline_4f-SketchLine_21r-SketchLine_20r"),
                            model.selection("FACE",
                                "Sketch_1/Face-SketchBSpline_1r-SketchLine_15r-SketchLine_14r"),
                            model.selection("FACE",
```

```
                                    "Sketch_1/Face-SketchBSpline_2f-SketchLine_17r-SketchLine_16r")]
ExtrusionCut_7 = model.addExtrusionCut(Part_2_doc, ExtrusionCut_7_objects_1,
    model.selection(),
                                        [model.selection("SOLID", "Box_1_1")])
ExtrusionCut_8 = model.addExtrusionCut(Part_2_doc, [model.selection("COMPOUND",
    "Sketch_4")], model.selection(), 0,
                                        "l*0.075", [model.selection("SOLID",
                                            "ExtrusionCut_1_1")])
Extrusion_1 = model.addExtrusion(Part_2_doc, [model.selection("FACE",
    "Sketch_4/Face-SketchCircle_1_2f")],
                                    model.selection(), "-l*0.005", "l*0.055",
                                        "Faces|Wires")
Extrusion_1.setName("Extrusion_3")
Extrusion_1.result().setName("Extrusion_3_1")
Extrusion_2 = model.addExtrusion(Part_2_doc, [model.selection("WIRE",
    "Sketch_4/Face-SketchCircle_2_2f_wire")],
                                    model.selection(), "-l*0.005", "l*0.055",
                                        "Faces|Wires")
Extrusion_2.setName("Extrusion_4")
Extrusion_2.result().setName("Extrusion_4_1")
Fillet_3 = model.addFillet(Part_2_doc, [
    model.selection("EDGE",
        "[Extrusion_3_1/Generated_Face&Sketch_4/SketchCircle_1_2][Extrusion_3_1/To_Face]"),
    model.selection("EDGE",
        "[Extrusion_3_1/Generated_Face&Sketch_4/SketchCircle_1_2][Extrusion_3_1/From_Face]")],
                            "0.01*l", keepSubResults=True)
Fillet_4 = model.addFillet(Part_2_doc, [
    model.selection("EDGE",
        "[Extrusion_4_1/Generated_Face&Sketch_4/SketchCircle_2_2][Extrusion_4_1/To_Face]"),
    model.selection("EDGE",
        "[Extrusion_4_1/Generated_Face&Sketch_4/SketchCircle_2_2][Extrusion_4_1/From_Face]")],
                            "0.01*l", keepSubResults=True)
Fillet_5 = model.addFillet(Part_2_doc, [model.selection("EDGE",
                                                        "[ExtrusionCut_2_1/Modified_Face&Sketch_2/Sket
                                         model.selection("EDGE",
                                                        "[ExtrusionCut_2_1/Modified_Face&Sketch_3/Sket
                            bigFilletRadius, keepSubResults=True)
Fillet_6_objects = [model.selection("FACE",
    "Fillet_3_1/MF:Fillet&Sketch_3/SketchBSpline_6"),
                    model.selection("FACE",
                        "Fillet_3_1/MF:Fillet&Sketch_4/SketchEllipse_2"),
                    model.selection("FACE",
                                    "(Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5)(Fillet_3_1/MF:Fil
                    model.selection("FACE",
                        "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_3"),
```

```
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_4"),
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&Box_1_1/Right"),
                        model.selection("FACE",
                            "Fillet_3_1/MF:Fillet&Sketch_4/SketchEllipse_1"),
                        model.selection("FACE",
                                        "(Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5)(ExtrusionCut_2_1/
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_1"),
                        model.selection("FACE",
                            "ExtrusionCut_2_1/Modified_Face&ExtrusionCut_2_1/From_Face_2"),
                        model.selection("EDGE",
                                        "[Fillet_3_1/MF:Fillet&Sketch_2/SketchBSpline_5][Fillet_3_1/MF:Fil
                        model.selection("EDGE",
                                        "[ExtrusionCut_2_1/Modified_Face&Sketch_1/SketchBSpline_4][Fillet_
#model.selection("EDGE",
#
    "[Fillet_3_1/MF:Fillet&Sketch_1/SketchBSpline_3][Fillet_3_1/MF:Fillet&Sketch_1/SketchLine_40]"),
#model.selection("EDGE",
#
    "[Fillet_3_1/MF:Fillet&Sketch_3/SketchBSpline_7][Fillet_3_1/MF:Fillet&Sketch_1/SketchLine_40]")]
    Fillet_6 = model.addFillet(Part_2_doc, Fillet_6_objects, filletRadius,
        keepSubResults=True)
    Symmetry_1_objects = [model.selection("SOLID", "Fillet_1_1"),
                          model.selection("SOLID", "Fillet_2_1"),
                          model.selection("SOLID", "Fillet_4_1")]
    Symmetry_1 = model.addSymmetry(Part_2_doc, Symmetry_1_objects, model.selection("FACE",
        "PartSet/XOZ"),
                                   keepOriginal=True, keepSubResults=True)
    Fuse_2 = model.addFuse(Part_2_doc,
                           [model.selection("SOLID", "Symmetry_1_3_1"),
                               model.selection("SOLID", "Symmetry_1_3_2")],
                           removeEdges=True, keepSubResults=True)


#
    ----------------------------------------------------------------------------------------------------
# Translation to match position in the wind tunnel
#
    ----------------------------------------------------------------------------------------------------
Translation_1 = model.addTranslation(Part_2_doc, [model.selection("COMPOUND",
    "Symmetry_1_1")], axis=model.selection("EDGE", "PartSet/OX"),
    distance="-TunnelPlacement", keepSubResults=True)
Translation_2 = model.addTranslation(Part_2_doc, [model.selection("COMPOUND",
```

```
    "Symmetry_1_2")], axis=model.selection("EDGE", "PartSet/OX"),
    distance="-TunnelPlacement", keepSubResults=True)
Translation_3_objects = [model.selection("SOLID", "Fuse_1_1"),
                         model.selection("SOLID", "Box_2_1"),
                         model.selection("SOLID", "Box_3_1"),
                         model.selection("SOLID", "Box_4_1"),
                         model.selection("SOLID", "Box_5_1"),
                         model.selection("SOLID", "Box_6_1")]
Translation_3 = model.addTranslation(Part_2_doc, Translation_3_objects,
    axis=model.selection("EDGE", "PartSet/OX"), distance="-TunnelPlacement",
    keepSubResults=True)

model.end()

Translation_3.results()[1].setTransparency(0.8)
Translation_3.results()[2].setTransparency(0.8)
Translation_3.results()[3].setTransparency(0.8)
Translation_3.results()[4].setTransparency(0.8)
Translation_3.results()[5].setTransparency(0.8)


Export_7 = model.exportToXAO(Part_2_doc, folder + "/XAO/RearWheels.xao",
    model.selection("COMPOUND", "Translation_1_1"), 'XAO')
Export_8 = model.exportToXAO(Part_2_doc, folder + "/XAO/FrontWheels.xao",
    model.selection("COMPOUND", "Translation_2_1"), 'XAO')
Export_9 = model.exportToXAO(Part_2_doc, folder + "/XAO/VehicleBody.xao",
    model.selection("COMPOUND", "Translation_3_1"), 'XAO')
Export_10 = model.exportToXAO(Part_2_doc, folder + "/XAO/VehicleRefinementBox4.xao",
    model.selection("SOLID", "Translation_3_2"), 'XAO')
Export_11 = model.exportToXAO(Part_2_doc, folder + "/XAO/OpenAirRefinement2.xao",
    model.selection("SOLID", "Translation_3_3"), 'XAO')
Export_12 = model.exportToXAO(Part_2_doc, folder + "/XAO/OpenAirRefinement1.xao",
    model.selection("SOLID", "Translation_3_4"), 'XAO')
Export_13 = model.exportToXAO(Part_2_doc, folder + "/XAO/OpenAirBoundaryBox.xao",
    model.selection("SOLID", "Translation_3_5"), 'XAO')
Export_14= model.exportToXAO(Part_2_doc, folder + "/XAO/VehicleRefinementBox5.xao",
    model.selection("SOLID", "Translation_3_6"), 'XAO')

model.do()
model.end()


#
    ------------------------------------------------------------------------------------------------------------
# Exporting STL Files at the specified resolution to the specified folder
```

```
#
    ------------------------------------------------------------------------------------------------------------
geompy = geomBuilder.New()

(imported, Tunnel, [], [], []) = geompy.ImportXAO(folder + "/XAO/Tunnel.xao")
(imported, NShear1, [], [], []) = geompy.ImportXAO(folder + "/XAO/NShear1.xao")
(imported, NShear2, [], [], []) = geompy.ImportXAO(folder + "/XAO/NShear2.xao")
(imported, NShear3, [], [], []) = geompy.ImportXAO(folder + "/XAO/NShear3.xao")
(imported, ImportantArea1, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/ImportantArea1.xao")
(imported, ImportantArea2, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/ImportantArea2.xao")
(imported, VehicleRefinementBox4, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/VehicleRefinementBox4.xao")
(imported, VehicleRefinementBox5, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/VehicleRefinementBox5.xao")
(imported, OpenAirRefinement2, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/OpenAirRefinement2.xao")
(imported, OpenAirRefinement1, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/OpenAirRefinement1.xao")
(imported, OpenAirBoundaryBox, [], [], []) = geompy.ImportXAO(folder +
    "/XAO/OpenAirBoundaryBox.xao")
(imported, VehicleBody, [], [], []) = geompy.ImportXAO(folder + "/XAO/VehicleBody.xao")
(imported, FrontWheels, [], [], []) = geompy.ImportXAO(folder + "/XAO/FrontWheels.xao")
(imported, RearWheels, [], [], []) = geompy.ImportXAO(folder + "/XAO/RearWheels.xao")

Inlet = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
geompy.UnionIDs(Inlet, [701])
Outlet = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
geompy.UnionIDs(Outlet, [875])
Walls = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
geompy.UnionIDs(Walls, [3, 69, 76, 83, 88, 93, 98, 105, 110, 117, 122, 129, 136, 141, 150,
    155, 162, 167, 172, 177, 182, 189, 194, 201, 206, 211, 216, 221, 228, 233, 238, 243,
    250, 257, 261, 268, 273, 278, 282, 296, 310, 314, 321, 326, 330, 346, 351, 358, 363,
    368, 373, 380, 385, 399, 406, 411, 416, 421, 426, 429, 440, 445, 452, 457, 460, 464,
    469, 472, 477, 481, 485, 490, 494, 501, 508, 511, 515, 518, 523, 526, 529, 536, 541,
    546, 551, 554, 557, 563, 566, 571, 577, 582, 585, 589, 591, 596, 601, 606, 613, 618,
    622, 627, 634, 639, 644, 649, 654, 657, 666, 669, 678, 683, 686, 691, 696, 712, 719,
    724, 728, 731, 735, 738, 743, 745, 748, 751, 753, 756, 759, 762, 764, 769, 774, 779,
    784, 789, 794, 801, 804, 807, 810, 817, 820, 823, 828, 834, 836, 839, 842, 847, 852,
    857, 862, 867, 872, 886, 888, 891, 896, 901, 904, 907, 910, 913])
WallsSlip = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
geompy.UnionIDs(WallsSlip, [857, 901, 627, 910, 779, 842, 794, 862, 784, 867, 907, 896,
    904, 891, 913, 888, 182, 421, 683, 691, 429, 440, 686, 696])
WallsNoSlip = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
```

```
geompy.UnionIDs(WallsNoSlip, [3, 69, 76, 83, 88, 93, 98, 105, 110, 117, 122, 129, 136, 141,
    150, 155, 162, 167, 172, 177, 189, 194, 201, 206, 211, 216, 221, 228, 233, 238, 243,
    250, 257, 261, 268, 273, 278, 282, 296, 310, 314, 321, 326, 330, 346, 351, 358, 363,
    368, 373, 380, 385, 399, 406, 411, 416, 426, 445, 452, 457, 460, 464, 469, 472, 477,
    481, 485, 490, 494, 501, 508, 511, 515, 518, 523, 526, 529, 536, 541, 546, 551, 554,
    557, 563, 566, 571, 577, 582, 585, 589, 591, 596, 601, 606, 613, 618, 622, 634, 639,
    644, 649, 654, 657, 666, 669, 678, 712, 719, 724, 728, 731, 735, 738, 743, 745, 748,
    751, 753, 756, 759, 762, 764, 769, 774, 789, 801, 804, 807, 810, 817, 820, 823, 828,
    834, 836, 839, 847, 852, 872, 886])
Fillets = geompy.CreateGroup(Tunnel, geompy.ShapeType["FACE"])
geompy.UnionIDs(Fillets, [872, 857, 907, 613, 913, 847, 622, 862, 691, 823, 696, 834, 891,
    779, 784, 904, 769, 363, 380, 801, 399, 155, 167, 177, 189, 426, 296, 452, 201, 657,
    666, 546, 591, 358, 551, 321, 117, 129, 105, 93, 83, 273, 511, 536, 724, 541, 606, 817,
    810, 839, 836, 440, 686, 421, 683, 172, 162, 406, 639, 649, 678, 719, 654, 457, 385,
    194, 206, 216, 150])

geompy.ExportSTL(Inlet, folder + "/STL/Inlet.stl", False, stlRefinement, True)
geompy.ExportSTL(Outlet, folder + "/STL/Outlet.stl", False, stlRefinement, True)
geompy.ExportSTL(Walls, folder + "/STL/Walls.stl", False, stlRefinement, True)
geompy.ExportSTL(WallsSlip, folder + "/STL/WallsSlip.stl", False, stlRefinement, True)
geompy.ExportSTL(WallsNoSlip, folder + "/STL/WallsNoSlip.stl", False, stlRefinement, True)
geompy.ExportSTL(Fillets, folder + "/STL/Fillets.stl", False, stlRefinement, True)
geompy.ExportSTL(NShear1, folder + "/STL/NShear1.stl", False, stlRefinement, True)
geompy.ExportSTL(NShear2, folder + "/STL/NShear2.stl", False, stlRefinement, True)
geompy.ExportSTL(NShear3, folder + "/STL/NShear3.stl", False, stlRefinement, True)
geompy.ExportSTL(ImportantArea1, folder + "/STL/ImportantArea1.stl", False, stlRefinement,
    True)
geompy.ExportSTL(ImportantArea2, folder + "/STL/ImportantArea2.stl", False, stlRefinement,
    True)
geompy.ExportSTL(VehicleRefinementBox4, folder + "/STL/VehicleRefinementBox4.stl", False,
    stlRefinement, True)
geompy.ExportSTL(VehicleRefinementBox5, folder + "/STL/VehicleRefinementBox5.stl", False,
    stlRefinement, True)
geompy.ExportSTL(OpenAirRefinement2, folder + "/STL/OpenAirRefinement2.stl", False,
    stlRefinement, True)
geompy.ExportSTL(OpenAirRefinement1, folder + "/STL/OpenAirRefinement1.stl", False,
    stlRefinement, True)
geompy.ExportSTL(OpenAirBoundaryBox, folder + "/STL/OpenAirBoundaryBox.stl", False,
    stlRefinement, True)
geompy.ExportSTL(VehicleBody, folder + "/STL/VehicleBody.stl", False, stlRefinement, True)
geompy.ExportSTL(FrontWheels, folder + "/STL/FrontWheels.stl", False, stlRefinement, True)
geompy.ExportSTL(RearWheels, folder + "/STL/RearWheels.stl", False, stlRefinement, True)

geompy.ExportSTL(VehicleBody, folder + "/STL/VehicleBodyASCII.stl", True, 0.0001, True)
geompy.ExportSTL(FrontWheels, folder + "/STL/FrontWheelsASCII.stl", True, 0.0001, True)
```

```
geompy.ExportSTL(RearWheels, folder + "/STL/RearWheelsASCII.stl", True, 0.0001, True)


shutil.rmtree(folder + "/XAO")

if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser()
```

B.6 Equations to compute Principal Components

$$PC_1 = 0.2887L + 0.2887\frac{W}{L} - 0.2887\frac{H}{L} - 0.2887\frac{W_N}{L} - 0.2887\frac{H_N}{L} - 0.2887\frac{D_N}{L} - 0.2887\frac{W_C}{L} - 0.2887\frac{H_C}{L} - 0.2887\frac{D_C}{L} + 0.2887\alpha + 0.2887\beta + 0.2887x$$

$$PC_2 = -0.1004l + 0.2809\frac{w}{l} + 0.4602\frac{h}{l} + 0.3660\frac{v}{l^3} - 0.0693\frac{o_f}{l} - 0.1935\frac{o_r}{l} + 0.4462\frac{h_f}{l} + 0.4381\frac{h_r}{l} + 0.3633\frac{r}{l}$$

$$PC_3 = 0.4851l - 0.4756\frac{w}{l} + 0.0608\frac{h}{l} + 0.0476\frac{v}{l^3} - 0.2812\frac{o_f}{l} - 0.5369\frac{o_r}{l} + 0.1433\frac{h_f}{l} + 0.1635\frac{h_r}{l} + 0.3434\frac{r}{l}$$

$$PC_4 = -0.4380l + 0.0407\frac{w}{l} - 0.0817\frac{h}{l} - 0.4883\frac{v}{l^3} + 0.6947\frac{o_f}{l} - 0.0620\frac{o_r}{l} + 0.2525\frac{h_f}{l} + 0.1840\frac{h_r}{l} + 0.1346\frac{r}{l}$$

B.7 PC values for new sampling points of parametric study

Configuration	PC1	PC2	PC3	PC4
config1	-2.0419	-0.9891	-2.9049	-0.2089
config2	1.9968	-1.7972	1.0589	-1.4341
config3	0.6542	1.3324	-1.4011	-1.0473
config4	-1.1666	2.3802	-1.0414	0.6251
config5	2.7491	2.9912	-2.2817	-1.2457
config6	-3.2000	-0.2815	-0.4242	0.4414
config7	1.9163	1.5032	-1.7828	0.1552
config8	1.0518	0.5688	-0.0152	0.9775
config9	-1.4853	-2.1464	0.3882	-1.8792
config10	-0.5259	0.2600	0.7935	-0.4430

Table B.3: Principal Component values for configurations of the parametric study

B.8 MATLAB code for the generation of geometry parameters for parametric study

```
clc; clear; close all
%% Calculate PCA coefficients

vehicle1 = [5.190, 0.3892, 0.3430, 0.04, 0.1836, 0.2241, 0.1524, 0.1536, 0.035];
vehicle2 = [4.670, 0.3919, 0.3148, 0.0307, 0.2088, 0.2126, 0.1409, 0.1420, 0.0218];
vehicle3 = [4.884, 0.3831, 0.3053, 0.0285, 0.2074, 0.2232, 0.1425, 0.1433, 0.03];
vehicle4 = [4.351, 0.4132, 0.3367, 0.0356, 0.1818, 0.1514, 0.1508, 0.1512, 0.026];
vehicle5 = [4.651, 0.3997, 0.3619, 0.0368, 0.2131, 0.2004, 0.1707, 0.1752, 0.0475];

tunnel1 = [20.42, 0.7299, 0.5337, 0.3399, 0.1909, 0.0609, 0.4258, 0.2657, 0.1878, 15, 2.5,
    1.5748];
tunnel2 = [17.58, 0.6940, 0.6143, 0.3982, 0.2617, 0.1991, 0.4437, 0.3185, 0.1991, 1, 1,
    -0.29];

parameters = [tunnel1, vehicle1;
              tunnel1, vehicle2;
              tunnel1, vehicle3;
              tunnel1, vehicle4;
              tunnel1, vehicle5;
              tunnel2, vehicle1;
              tunnel2, vehicle2;
              tunnel2, vehicle3;
              tunnel2, vehicle4;
              tunnel2, vehicle5];
standardizedData = zscore(parameters);
[coeff, ~, ~, ~, explained] = pca(standardizedData);
clear tunnel* vehicle*

cumulativeVariance = cumsum(explained(1:4));
figure('Position', [0, 0, 5000, 3000])
bar(explained(1:4))
hold on
plot(cumulativeVariance)

% Add marker at the last cumulative point
xCumulative = length(cumulativeVariance);
yCumulative = cumulativeVariance(end);
plot(xCumulative, yCumulative, 'ro', 'MarkerSize', 6, 'MarkerFaceColor', 'r')
text(xCumulative, yCumulative - 4, sprintf('%.1f%%', yCumulative), 'HorizontalAlignment',
    'left', 'Color', 'r')
```

```
grid on

xlabel('Principal Component');
ylabel('- [%]');
lgd = legend('Explained Variance', 'Cumulative Variance', 'Location', 'northwest');
fontsize(lgd, 20, 'points')
fontsize(gca, 20, 'points')
meanDataset = mean(parameters);
stdDataset = std(parameters);

PCs = computePC(parameters, meanDataset, stdDataset, coeff);
rangePC1 = [min(PCs(:, 1)), max(PCs(:, 1))];
rangePC2 = [min(PCs(:, 2)), max(PCs(:, 2))];
rangePC3 = [min(PCs(:, 3)), max(PCs(:, 3))];
rangePC4 = [min(PCs(:, 4)), max(PCs(:, 4))];

rangePCs = [rangePC1; rangePC2; rangePC3; rangePC4];
%% Calculate new PC points from LHS

N = 10;

% Set random number generator to a known value to ensure same results everytime
rng default;
LHS_samples = lhsdesign(N, size(rangePCs, 1));

scaledLHS = zeros(size(LHS_samples));
for i = 1:size(rangePCs, 1)
    minPCs = rangePCs(i, 1);
    maxPCs = rangePCs(i, 2);
    scaledLHS(:, i) = LHS_samples(:, i)*(maxPCs - minPCs) + minPCs;
end
clear minPCs maxPCs i
%%
%

PC1 = scaledLHS(:, 1);
PC2 = scaledLHS(:, 2);
PC3 = scaledLHS(:, 3);

% Define the range for x and z
[y, z] = meshgrid(-5:0.5:5, -5:0.5:5);

% Choose a fixed x-value
xPlane1 = rangePC1(1);
x1 = xPlane1*ones(size(y));
```

```
xPlane2 = rangePC1(2);
x2 = xPlane2*ones(size(y));

figure;
plot3(PCs(1:5, 1), PCs(1:5, 2), PCs(1:5, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
grid on;
hold on
plot3(PCs(6:end, 1), PCs(6:end, 2), PCs(6:end, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor',
    'g')
plot3(PC1, PC2, PC3, '*', 'MarkerSize', 8);
surf(x1, y, z, 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'FaceColor', [0.5 0.5 0.5]);
surf(x2, y, z, 'FaceAlpha', 0.1, 'EdgeColor', 'none', 'FaceColor', [0.5 0.5 0.5]);

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
xlim(rangePC1)
ylim(rangePC2)
zlim(rangePC3)
title('LHS samples in the PC domain');
legend('Tunnel 1 data', 'Tunnel 2 data', 'LHS samples', 'Location', 'northwest')
figure('Position', [0, 0, 5000, 1000]);
t = tiledlayout(1, 3);

nexttile;
plot3(PCs(1:5, 1), PCs(1:5, 2), PCs(1:5, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
grid on;
hold on
plot3(PCs(6:end, 1), PCs(6:end, 2), PCs(6:end, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor',
    'g')
plot3(PC1, PC2, PC3, '*', 'MarkerSize', 8);
view(0, 90)

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
xlim(rangePC1)
ylim(rangePC2)
zlim(rangePC3)
title('XY view');
fontsize(gca, 15, 'points')

nexttile;
plot3(PCs(1:5, 1), PCs(1:5, 2), PCs(1:5, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
grid on;
```

```
hold on
plot3(PCs(6:end, 1), PCs(6:end, 2), PCs(6:end, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor',
    'g')
plot3(PC1, PC2, PC3, '*', 'MarkerSize', 8);
view(90, 0)

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
xlim(rangePC1)
ylim(rangePC2)
zlim(rangePC3)
title('YZ view');
fontsize(gca, 15, 'points')
legend('Tunnel 1 data', 'Tunnel 2 data', 'LHS samples', 'Location', 'southoutside',
    'Orientation', 'horizontal')

nexttile;
plot3(PCs(1:5, 1), PCs(1:5, 2), PCs(1:5, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
grid on;
hold on
plot3(PCs(6:end, 1), PCs(6:end, 2), PCs(6:end, 3), 'o', 'MarkerSize', 8, 'MarkerFaceColor',
    'g')
plot3(PC1, PC2, PC3, '*', 'MarkerSize', 8);
view(0, 0)

xlabel('PC1');
ylabel('PC2');
zlabel('PC3');
xlim(rangePC1)
ylim(rangePC2)
zlim(rangePC3)
title('XZ view');
fontsize(gca, 15, 'points')
%% Set ranges for tunnel and vehicle geometries
% _For tunnel..._

TunnelRanges.L = [17, 21];
TunnelRanges.W = [0.65, 0.8];
TunnelRanges.H = [0.5, 0.7];
TunnelRanges.W_n = [0.3, 0.5];
TunnelRanges.H_n = [0.2, 0.3];
TunnelRanges.D_n = [0.05, 0.2];
TunnelRanges.W_c = [0.4, 0.5];
TunnelRanges.H_c = [0.25, 0.35];
```

```
TunnelRanges.D_c = [0.1, 0.2];
TunnelRanges.alpha = [10, 20];
TunnelRanges.beta = [1, 5];
TunnelRanges.x = [-1/4, 1/4].*TunnelRanges.L;
%%
% _For vehicle..._

VehicleRanges.l = [3.8, 6];
VehicleRanges.w = [0.3, 0.43];
VehicleRanges.h = [0.28, 0.35];
VehicleRanges.v = [0.025, 0.035];
VehicleRanges.o_f = [0.17, 0.23];
VehicleRanges.o_r = [0.17, 0.23];
VehicleRanges.h_f = [0.14, 0.16];
VehicleRanges.h_r = [0.14, 0.16];
VehicleRanges.r = [0.02, 0.04];
%% Compute new parameters corresponding to LHS samples

lowerBoundary = [TunnelRanges.L(1), TunnelRanges.W(1), TunnelRanges.H(1),
    TunnelRanges.W_n(1), TunnelRanges.H_n(1), TunnelRanges.D_n(1), TunnelRanges.W_c(1),
    TunnelRanges.H_c(1), TunnelRanges.D_c(1), TunnelRanges.alpha(1), TunnelRanges.beta(1),
    TunnelRanges.x(1),...
                VehicleRanges.l(1), VehicleRanges.w(1), VehicleRanges.h(1),
                    VehicleRanges.v(1), VehicleRanges.o_f(1), VehicleRanges.o_r(1),
                    VehicleRanges.h_f(1), VehicleRanges.h_r(1), VehicleRanges.r(1)];
upperBoundary = [TunnelRanges.L(2), TunnelRanges.W(2), TunnelRanges.H(2),
    TunnelRanges.W_n(2), TunnelRanges.H_n(2), TunnelRanges.D_n(2), TunnelRanges.W_c(2),
    TunnelRanges.H_c(2), TunnelRanges.D_c(2), TunnelRanges.alpha(2), TunnelRanges.beta(2),
    TunnelRanges.x(2),...
                VehicleRanges.l(2), VehicleRanges.w(2), VehicleRanges.h(2),
                    VehicleRanges.v(2), VehicleRanges.o_f(2), VehicleRanges.o_r(2),
                    VehicleRanges.h_f(2), VehicleRanges.h_r(2), VehicleRanges.r(2)];
options = optimoptions(@lsqnonlin, 'Algorithm', 'levenberg-marquardt', 'Display', 'none',
    'MaxFunctionEvaluations', 3000);

parametersMatrix = zeros(N, length(coeff));
JMatrix = zeros(N, 1);
for i = 1:N
    desiredPC = scaledLHS(i, :);
    x0 = lowerBoundary + (upperBoundary - lowerBoundary)*rand();

    % Optimization
    [optimizedParameters, JMin] = lsqnonlin(@(x) objectiveFunction(x, desiredPC,
        meanDataset, stdDataset, coeff), x0, lowerBoundary, upperBoundary, options);
    parametersMatrix(i, :) = round(optimizedParameters, 4);
```

```
    JMatrix(i, :) = JMin;
end

tunnelParameters = parametersMatrix(:, 1:12);
vehicleParameters = parametersMatrix(:, 13:end);

outputFile = fopen('modelParameters.txt', 'w');
for i = 1:size(tunnelParameters, 1)
    tunnel = tunnelParameters(i, :);
    vehicle = vehicleParameters(i, :);

    % Join in a single string with commas in between
    tunnelStr = join(string(tunnel), ', ');
    vehicleStr = join(string(vehicle), ', ');

    fprintf(outputFile, '---%d---\n', i);
    fprintf(outputFile, 'TunnelParam = [%s]\n', tunnelStr);
    fprintf(outputFile, 'VehicleParam = [%s]\n\n', vehicleStr);
end
fclose(outputFile);

writematrix(tunnelParameters, 'TunnelParam.xls');
writematrix(vehicleParameters, 'VehicleParam.xls');
%% Functions

function PC = computePC(parameters, meanValue, stdValue, coeffPCA)
    standard = (parameters - meanValue)./stdValue;

    PC1 = standard * coeffPCA(:, 1);
    PC2 = standard * coeffPCA(:, 2);
    PC3 = standard * coeffPCA(:, 3);
    PC4 = standard * coeffPCA(:, 4);

    PC = [PC1, PC2, PC3, PC4];
end

function J = objectiveFunction(parameters, desiredPC, meanValue, stdValue, coeffPCA)
    modelPC = computePC(parameters, meanValue, stdValue, coeffPCA);
    J = desiredPC - modelPC;
end
```

C. Appendix C

C.1 MATLAB script for data analysis

```
close all
clear
clc

addpath("Tunnel\config1\")
%% Import _.log_ files

logCoefficients = importdata("001_config1_TUNNEL.out");
logCoefficients = logCoefficients(:, [1 2]);
%% Manage data to remove redundancy
% _Single file_

CdData = removeDuplicates(logCoefficients);

iteration = CdData(:, 1);
Cd = CdData(:, 2);
%% Analysis
% Evaluate $C_{d, mean}$ over 10000 iterations

CdMean = movmean(Cd, [9999, 0]);
%% Convergence based on adimensional difference

transientCriteria = 0.015;
convergenceCriteria = 1e-4;

% Calculate new metric based on min-to-max difference of CdMean
maxCdMean = movmax(CdMean, [9999, 0]);
minCdMean = movmin(CdMean, [9999, 0]);
diffCdMean = maxCdMean - minCdMean;
adimensionalCd = diffCdMean./CdMean;

% Calculate moving min and max of the new metric to assess convergence
maxDiffCdMean = movmax(diffCdMean, [2999, 0]);
minDiffCdMean = movmin(diffCdMean, [2999, 0]);
convergenceMetric = maxDiffCdMean - minDiffCdMean;
```

```
index = 0;
for i = 100:length(diffCdMean)
    if (adimensionalCd(i) < transientCriteria) && (convergenceMetric(i) <
        convergenceCriteria)
        fprintf('Convergence reached at iteration %d with value %.4f\n', min(iteration) +
            i, CdMean(i));
        index = i;
        break;
    end
end
results = {'config1', round(CdMean(index), 4), min(iteration) + index};
fileName = 'Results.xlsx';
writecell(results, fileName, 'Sheet', 'Tunnel', 'Range', 'A2');

figure('Position', [0, 0, 5000, 3000])
plot(iteration, adimensionalCd)
grid minor
xlabel('iteration')
ylabel('\Delta{C_d}/C_{d,mean}')
xlim([min(iteration), max(iteration)])
hold on
plot(min(iteration) + index, adimensionalCd(index), 'Color', 'g', 'Marker', '.',
    'MarkerSize', 15)
title('001\_config1\_TUNNEL')
%% Comparison

figure('Position', [0, 0, 2000, 700])
plot(iteration, Cd)
hold on, grid minor
plot(iteration, CdMean, 'LineWidth', 2)
xlabel('iteration')
xlim([min(iteration), max(iteration)])
ylabel('C_d')
ylim([0.1 0.4])
title('Drag coefficient vs. iteration for 001\_config1\_TUNNEL')
hold on
plot(min(iteration) + index, CdMean(index), 'Color', 'g', 'Marker', '.', 'MarkerSize', 15)
legend('C_d', 'C_{d, mean}', 'Convergence point', 'Location', 'northeast')
CdStart = CdMean(1, :);
fprintf('Cd at the start = %.4f', CdStart)
%% Functions

function newData = removeDuplicates(oldData)
    newData = [];
    duplicates = [];
```

```
    for i = 1:size(oldData, 1)
        value = oldData(i, 1);
        if not(ismember(value, duplicates))
            newData = [newData; oldData(i, :)];
            duplicates = [duplicates, value];
        end
    end
end
```

C.2 Terms of dynamic pressure correction

ε_S refers to **jet expansion** correction:

$$\varepsilon_S = \tau \frac{v^{0.5}}{l} \frac{A^{1.5}}{A^*} \tag{C.1}$$

where $\tau = 0.36 \left(\frac{W}{H} + \frac{H}{W} \right)$ = tunnel shape factor

v = vehicle volume

l = vehicle length

A = vehicle frontal area

$A^* = \frac{A_N}{1 + \varepsilon_{QN}}$

A_N = nozzle cross-sectional area

ε_C refers to **collector blockage** correction:

$$\varepsilon_C = \frac{\varepsilon_W R_C^3}{\left[(L - x_M)^2 + R_C^2 \right]^{1.5}} \tag{C.2}$$

where $\varepsilon_W = \frac{A}{A_C} \left(\frac{C_d}{4} + 0.41 \right)$ = wake blockage factor

$R_C = \sqrt{\frac{2A_C}{\pi}}$

A_C = collector cross-sectional area

L = test section length

x_M = distance from nozzle to vehicle center

ε_N refers to **nozzle blockage** correction:

$$\varepsilon_N = \frac{\varepsilon_{QN} R_N^3}{(x_M^2 + R_N^2)^{1.5}} \tag{C.3}$$

$$\text{where } R_N = \sqrt{\frac{2A_C}{\pi}}$$

ε_{QN} refers to **blockage correction for nozzle-method**:

$$\varepsilon_{QN} = \frac{A}{2A_N}\left(\frac{1 - x_s}{\sqrt{x_s^2 + R_N^2}}\right) \tag{C.4}$$

$$\text{where } x_s = x_M - \frac{l}{2} + \sqrt{\frac{A}{2\pi}} = \text{distance from vehicle center to source point}$$

ε_{QP} refers to **blockage correction for plenum-method**:

$$\varepsilon_{QP} = \frac{A}{2\pi}\left(\frac{x_s}{(x_s^2 + R_N^2)^{1.5}}\right) \tag{C.5}$$

C.3 Drag coefficient values for parametric study

Table C.1: Values of drag coefficient for each configuration in the different domains

Configuration	$\mathbf{C_{d,tunnel}}$	$\mathbf{C_{d,q}}$	$\mathbf{C_{d,open}}$
config1	0.1583	0.1769	0.2392
config2	0.0752	0.1996	0.3204
config3	0.2125	0.1968	0.2561
config4	0.1973	0.1939	0.1953
config5	0.2641	0.2419	0.2887
config6	0.2092	0.2052	0.2384
config7	0.2317	0.2123	0.2719
config8	0.0535	0.2112	0.3116
config9	0.0029	0.1753	0.2882
config10	0.1788	0.2158	0.2981

C.4 ΔC_d values for of the parametric dataset

Table C.2: ΔC_d for configurations of the parametric study

Configuration	$\mathbf{\Delta C_d}$
config1	0.0556
config2	0.1055
config3	0.0525
config4	0.0013
config5	0.0408
config6	0.0295
config7	0.0524
config8	0.0877
config9	0.0998
config10	0.0720

C.5 ΔC_d values for of the experimental dataset

Table C.3: ΔC_d for experimental dataset

Configuration	**Tunnel 1**	**Tunnel 2**
vehicle 1	0.0872	0.0518
vehicle 2	0.0991	0.0583
vehicle 3	0.1049	0.0670
vehicle 4	0.0608	0.0234
vehicle 5	0.0601	0.0145

C.6 ΔC_d between tunnels for different correction methods

Table C.4: $C_{d,q,T1} - C_{d,q,T2}$ values for different correction methods

Configuration	**Classic**	**TM**	**RBF$_{10+10}$**	**RBF$_{30}$**	**RBF$_{50}$**	**RBF$_{10}$**
vehicle 1	0.0161	0.0022	0.0145	0.0063	0.0134	0.0516
vehicle 2	0.0178	−0.0006	0.0214	0.0024	0.0149	0.0586
vehicle 3	0.0084	−0.0045	0.0126	−0.0060	0.007	0.0453
vehicle 4	0.0323	0.0293	0.0231	0.0248	0.0226	0.0697
vehicle 5	0.0372	0.0139	0.0144	0.0179	0.0132	0.0827
μ	0.0224	0.0081	0.0172	0.0091	0.0142	0.0616
σ	0.0120	0.0137	0.0047	0.0123	0.0056	0.0149

C.7 ΔC_d between tunnels combining correction methods

Table C.5: Comparison of $C_{d,q,T1} - C_{d,q,T2}$ by combining the different RBFs

Configuration	$\mathbf{RBF_{10} + (RBF_{50} - RBF_{30})}$	$\mathbf{RBF_{10} + RBF_{10+10}}$
vehicle 1	0.0587	0.0661
vehicle 2	0.0711	0.0800
vehicle 3	0.0583	0.0579
vehicle 4	0.0675	0.0928
vehicle 5	0.0780	0.0971
μ	0.0667	0.0788
σ	0.0084	0.0168

VITA AUCTORIS

NAME:	Philippe Pession
PLACE OF BIRTH:	Aosta (AO)
YEAR OF BIRTH:	2001
EDUCATION:	Bachelor Degree in Automotive Engineering at Politecnico di Torino, 2020-2023
	Master Degree in Automotive Engineering at Politecnico di Torino, 2023-2025
	M.A.Sc in Automotive Engineering at University of Windsor, 2024-2025