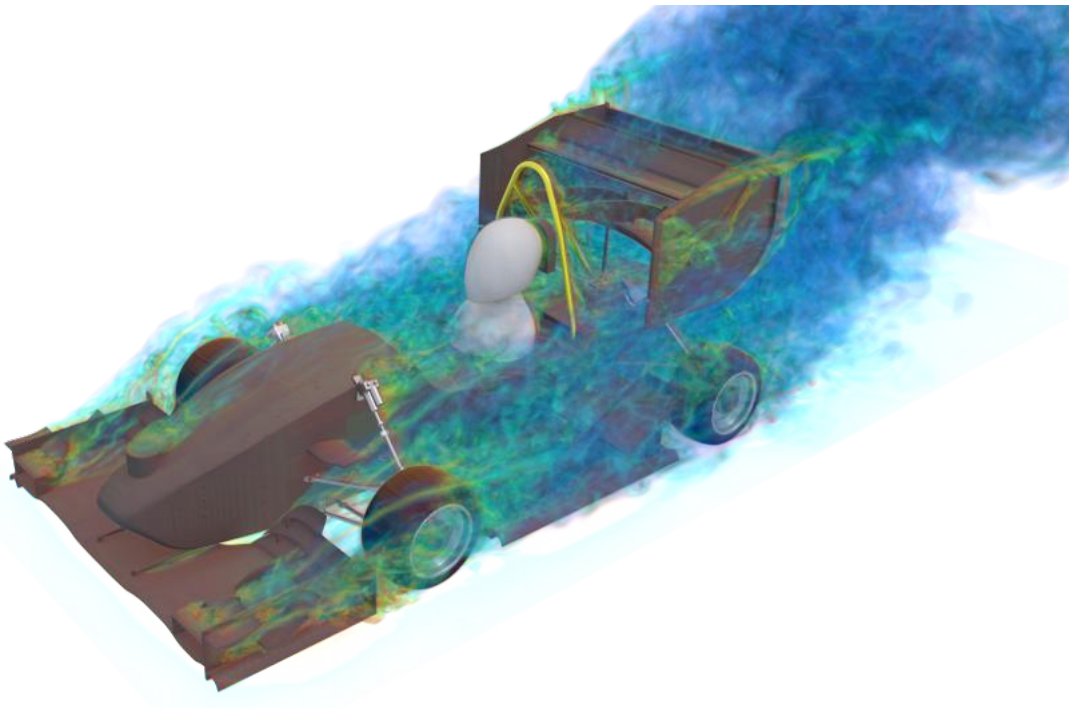
CHALMERS
UNIVERSITY OF TECHNOLOGY



Transient Analysis of Active Aerodynamics for a Formula Student Front Wing

Master's thesis in Mobility Engineering

PANOS PAPAKOSTAS

DEPARTMENT OF MECHANICS AND MARITIME SCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Transient Analysis of Active Aerodynamics for a Formula Student Front Wing

PANOS PAPAKOSTAS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Science
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2026

Transient Analysis of Active Aerodynamics for a Formula Student Front Wing
PANOS PAPAKOSTAS

Supervisor: Alexey Vdovin, Division of Vehicle Engineering and Autonomous Systems
Examiner: Alexey Vdovin, Division of Vehicle Engineering and Autonomous Systems

Master's Thesis 2026
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Turbulence Illustration of the CFS26 car.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Transient Analysis of Active Aerodynamics for a Formula Student Front Wing
PANOS PAPAKOSTAS
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This research aims to design modifications for a Formula Student car so that it has active aerodynamic devices focusing on the front wing. The modifications will be simulated using transient flow physics in computational fluid dynamics simulations employing Detached Eddy Simulation, focusing on moving geometry effects at different actuation and vehicle speeds. The aim is to quantify the time lag between mechanical movement and the stabilization of aerodynamic forces, identify the presence of unstable vortex shedding generated during the transition and to establish the relationship between actuation speed and flow efficiency to determine a safe working window for aerodynamic stability while reducing drag as much as possible. The simulations were carried out across five DRS configurations, leading to a drag reduction of up to 5.5% while maintaining stable transient behavior during actuation.

Keywords: aerodynamics, transient aerodynamics, formula student, drag reduction system, computational fluid dynamics.

Acknowledgements

I would like to thank my supervisor, Alexey Vdovin, for the guidance throughout the thesis and for giving me the needed resources that made this work possible.
To my family, thank you for the constant support and believing in me throughout this degree and standing by me through the toughest stretches of this thesis.
To my friends back home thank you for staying close despite the distance and for always being there when I need a break. And to the friends I made along the way here at Chalmers as well as in CFS, thank you for all the memories made along the way.

Panos Papakostas, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AoA	Angle of Attack
BCD	Bounded Central Differencing
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy number
CFS	Chalmers Formula Student
DES	Detached Eddy Simulation
DRS	Drag Reduction System
FS	Formula Student
IDDES	Improved Delayed Detached Eddy Simulation
LES	Large Eddy Simulation
RANS	Reynolds-Averaged Navier-Stokes
SIMPLEC	Semi-Implicit Method for Pressure Linked Equations Consistent
SST	Shear Stress Transport

Nomenclature

Below is the nomenclature parameters and abbreviations that have been used throughout this thesis.

Parameters

AR	Aspect ratio
b	Distance from rear axle to centre of pressure
c	Chord length
c_{ref}	Reference chord length
C_D	Drag coefficient
C_L	Lift (downforce) coefficient
C_M	Pitching moment coefficient
C_p	Pressure coefficient
$C_{p,tot}$	Total pressure coefficient
D	Drag force
e	Span efficiency factor
F_D	Aerodynamic drag force
F_z	Vertical aerodynamic force (downforce)
$F_{z,front}$	Front axle aerodynamic load
$F_{z,rear}$	Rear axle aerodynamic load
$F_{z,total}$	Total aerodynamic downforce
f	Vortex shedding frequency
H	Boundary layer shape factor
h	Ride height
h_{cp}	Height of centre of pressure above ground
k	Turbulent kinetic energy
L	Lift force

L_{WB}	Vehicle wheelbase
p	Static pressure
p_∞	Freestream static pressure
Re	Reynolds number
S_{ref}	Reference area
St	Strouhal number
t	Time
t_{start}	DRS actuation start time
t_{end}	DRS actuation end time
U_∞	Freestream velocity
U_e	Boundary layer edge velocity
u, v, w	Velocity components
W	Vehicle weight
y^+	Non-dimensional wall distance
δ	Boundary layer thickness
δ^*	Displacement thickness
δ_f	Flap deflection angle
θ	Momentum thickness
θ_{max}	Maximum flap deflection angle
Γ	Circulation
μ	Dynamic viscosity
ν	Kinematic viscosity
ρ	Fluid density
τ	Viscous stress tensor
ω	DRS flap angular velocity
$\boldsymbol{\omega}$	Vorticity vector
Φ	General flow variable

Contents

List of Figures

List of Tables

1
Introduction

Formula Student (FS) is a student engineering competition in which universities compete with formula style cars designed and built by the student. Each competition features both static and dynamic events.
By adding a Drag Reduction System (DRS) to the vehicle, points can be gained for the team in both static events like engineering design event which challenges the design that the students came up with and give points depending on the design, the next part of the competition is the dynamic events which consist of acceleration, skidpad, autocross and endurance (trackdrive for autonomous) testing the performance and efficiency of the car on track.

1.1 Background

Vehicle aerodynamics play a crucial role in improving lap time and improving the performance of the vehicle, by increasing downforce while minimizing drag. One of the main aerodynamic components is the front wing, generating about one third of the downforce and has a major effect on the aerodynamic balance and the airflow downstream.
In recent years, active aerodynamics have gained attention in motorsports and automotive applications due to their potential to optimize the aerodynamic characteristics of the vehicle based on the driving conditions. By dynamically altering the wings angle of attack it is possible to reduce drag on straight while keeping the high downforce setup for cornering. However, most active aerodynamics studies are performed using steady-state simulations which neglect time-dependent effects such as actuator motion and transient flow behavior.
A transient aerodynamic analysis provides insight into unsteady flow phenomena and the dynamic response of the active system. This is essential in designing a reliable and efficient adaptive system. Therefore this thesis investigates the transient aerodynamic behavior of a modified Formula Student front wing to better understand its potential.

1.2 Aim

The aim of this study is to investigate using commercial CFD software, the transient aerodynamic response of an actuated Formula Student front wing. With particular focus on time dependent effects such as aerodynamic hysteresis, actuation induced

time lag and unsteady flow behavior. In order to evaluate the feasibility and performance implications of active aerodynamic system in Formula Student applications.

1.3 Limitations

This study is subject to several limitations. Detached Eddy Simulation (DES) will be used, which offers improved prediction of unsteady flow behavior compared to URANS approaches; it still relies on turbulence modeling approximations, and the transition between RANS and LES region may affect accuracy.
Another limitation is the computational power that is needed for a simulation of a full car- moving geometry - DES simulation, as well as the time for a simulation converge.
In addition, the investigation is restricted to a limited range of operating conditions such as vehicle speed and DRS rotation speed. For example yaw angle as well as pitch changes are not considered.
Finally, validation of the results will not be possible as no experimental data are available, thus the results will be based on numerical simulations which produces uncertainties associated with modeling assumptions and settings.

2

Theory

2.1 General Fluid Mechanics

This section establishes the fundamental principles of fluid mechanics that govern the flow around aerodynamic surfaces. These conservation laws form the foundation for both analytical understanding and computational simulations.

2.1.1 Governing Equations

The flow field around the FS car is described by the fundamental conservation laws of fluid mechanics. For the low-speed regime ($M < 0.3$), the incompressible flow assumption is valid [1].

2.1.1.1 Conservation of Mass (Continuity Equation)

The conservation of mass can be understood by considering a fixed control volume in the flow field. As illustrated in Figure 2.1, mass entering through one face must exit through other faces, with no accumulation for steady, incompressible flow.

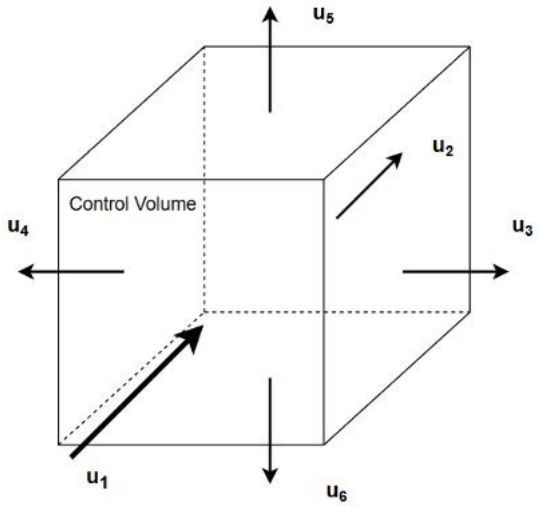


Figure 2.1: Control Volume for Deriving the Continuity Equation.

The integral form of mass conservation states [2]:

$$\frac{\partial}{\partial t}\int_{CV}\rho\,dV+\int_{CS}\rho\mathbf{u}\cdot\mathbf{n}\,dA=0 \tag{2.1}$$

where CV is the control volume, CS is its bounding surface, and $\mathbf{n}$ is the outward normal. For incompressible flow (ρ = constant) and applying the divergence theorem, this reduces to the differential form:

$$\nabla\cdot\mathbf{u}=0 \tag{2.2}$$

This states that the velocity field is divergence-free - the net volumetric flux through any point is zero.

2.1.1.2 Conservation of Momentum (Navier-Stokes Equations)

The conservation of momentum is governed by the incompressible Navier-Stokes equations [1]:

$$\frac{\partial\mathbf{u}}{\partial t}+(\mathbf{u}\cdot\nabla)\mathbf{u}=-\frac{1}{\rho}\nabla p+\nu\nabla^2\mathbf{u} \tag{2.3}$$

where:

- $\mathbf{u}=(u,v,w)$ is the velocity vector
- t is time
- ρ is the fluid density
- p is the pressure
- $\nu=\mu/\rho$ is the kinematic viscosity

The left-hand side represents the material derivative of velocity (total acceleration), while the right-hand side contains pressure gradient forces and viscous forces. Each term has physical significance:

- $\frac{\partial\mathbf{u}}{\partial t}$: Local (unsteady) acceleration
- $(\mathbf{u}\cdot\nabla)\mathbf{u}$: Convective acceleration
- $-\frac{1}{\rho}\nabla p$: Pressure gradient force per unit mass
- $\nu\nabla^2\mathbf{u}$: Viscous diffusion (friction) force per unit mass

2.1.1.3 Non-Dimensional Parameters

The flow behavior is characterized by non-dimensional parameters. The Reynolds number represents the ratio of inertial to viscous forces [1]:

$$Re=\frac{U_\infty c}{\nu}=\frac{\rho U_\infty c}{\mu} \tag{2.4}$$

where U_∞ is the freestream velocity, c is the characteristic length (chord), and μ is the dynamic viscosity.
This Reynolds number range indicates turbulent flow over most of the wing surface, with possible laminar regions near leading edges [3].

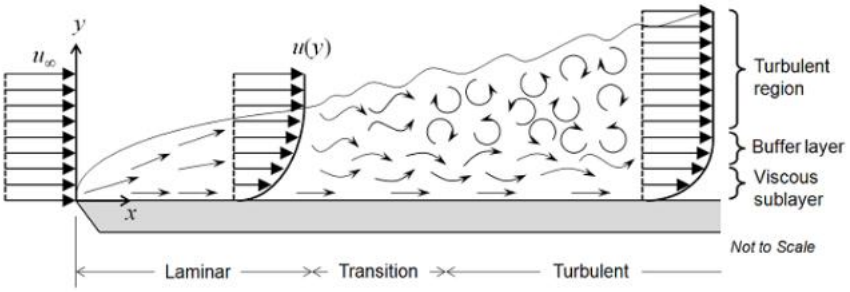


Figure 2.2: Boundary Layer Transition.

2.1.2 Boundary Layer Theory

The boundary layer is the thin region adjacent to a solid surface where viscous effects are significant [3]. Understanding boundary layer behavior is critical for accurate aerodynamic predictions.

2.1.2.1 Boundary Layer Equations

Within the boundary layer, the flow satisfies simplified equations derived from the Navier-Stokes equations under the assumption that $\delta \ll L$ (boundary layer thickness much smaller than characteristic length) [3]:
For 2D steady flow:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2 u}{\partial y^2} \tag{2.5}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.6}$$

where x is the streamwise coordinate along the surface, y is the normal coordinate, and u, v are velocity components. The pressure gradient dp/dx is determined by the inviscid outer flow:

$$\frac{dp}{dx} = -\rho U_e \frac{dU_e}{dx} \tag{2.7}$$

where $U_e(x)$ is the velocity at the boundary layer edge.

2.1.2.2 Boundary Layer Thickness Parameters

Several thickness measures characterize the boundary layer [1, 3]:
Boundary layer thickness δ: Distance from wall where $u = 0.99U_e$
Displacement thickness δ^*: Outward displacement of streamlines due to viscous effects:

$$\delta^* = \int_0^\infty \left(1 - \frac{u}{U_e}\right) dy \tag{2.8}$$

Momentum thickness θ: Momentum deficit in the boundary layer:

$$\theta = \int_0^\infty \frac{u}{U_e}\left(1 - \frac{u}{U_e}\right) dy \tag{2.9}$$

Shape factor H: Ratio indicating boundary layer health:

$$H = \frac{\delta^*}{\theta} \tag{2.10}$$

The shape factor values: $H \approx 2.6$ (laminar), $H \approx 1.3 - 1.4$ (turbulent), $H > 2.4$ (incipient separation) [3].

2.1.2.3 Flow Separation

Flow separation occurs when the boundary layer detaches from the surface due to adverse pressure gradient ($dp/dx > 0$) [3]. At the separation point:

$$\left(\frac{\partial u}{\partial y}\right)_{y=0} = 0 \tag{2.11}$$

Consequences of separation:

- Dramatic increase in pressure drag (form drag)
- Loss of lift/downforce
- Unsteady vortex shedding
- Increased turbulence levels

For race car wings, separation during DRS actuation can significantly degrade performance. Turbulent boundary layers resist separation better than laminar layers [1].

2.1.3 Vortex Dynamics

2.1.3.1 Vorticity and Circulation

Vorticity is a measure of local fluid rotation [4]:

$$\boldsymbol{\omega} = \nabla \times \mathbf{u} \tag{2.12}$$

Circulation around a closed contour C is:

$$\Gamma = \oint_C \mathbf{u} \cdot d\mathbf{l} \tag{2.13}$$

By Helmholtz's vortex theorems:

- Vortex filament strength is constant along its length
- Vortex filaments cannot end in the fluid (must form closed loops)
- Vortex lines move with the fluid

2.1.3.2 Vortex Shedding

For bluff bodies and at high angles of attack, periodic vortex shedding occurs with characteristic Strouhal number [4]:

$$St = \frac{fD}{U_\infty} \tag{2.14}$$

where f is the shedding frequency and D is a characteristic dimension. Understanding vortex dynamics is crucial for predicting transient loads during DRS actuation, when rapid geometry changes trigger vortex formation and shedding [5].

2.2 Wing Aerodynamics

This section applies fluid mechanics principles to the specific case of wings, with emphasis on multi-element configurations, ground effect, and transient phenomena relevant to Formula Student front wings with DRS.

2.2.1 Lift and Drag Generation

2.2.1.1 Aerodynamic Force Integration

Aerodynamic forces arise from pressure and shear stress distributions over the wing surface [1, 6]:

$$\mathbf{F} = -\int_S p\mathbf{n}\, dS + \int_S \boldsymbol{\tau} \cdot \mathbf{n}\, dS \tag{2.15}$$

where S is the wing surface, $\mathbf{n}$ is the outward normal, p is pressure, and $\boldsymbol{\tau}$ is the viscous stress tensor. The first integral (pressure forces) dominates lift, while the second (viscous forces) contributes mainly to friction drag.

2.2.1.2 Force Coefficients

Forces are non-dimensionalized:

$$C_L = \frac{L}{\frac{1}{2}\rho U_\infty^2 S_{ref}}, \quad C_D = \frac{D}{\frac{1}{2}\rho U_\infty^2 S_{ref}} \tag{2.16}$$

where L is the lift force, D is the drag force, ρ is the air density, U_∞ is the freestream velocity, and S_{ref} is the reference area which in this case is the projected frontal area of the vehicle.
For race cars the wings are used to generate downforce, hence C_L is negative by convention.

2.2.1.3 Pressure Distribution

The pressure coefficient describes the pressure field:

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2} \tag{2.17}$$

where C_p is the pressure coefficient, p is the local static pressure, and p_∞ is the freestream static pressure.
The lift on a wing section relates to circulation through the Kutta-Joukowski theorem [1]:

2.2.2 Drag Decomposition

Total drag comprises pressure and friction components [1]:

$$D_{total} = D_{pressure} + D_{friction} \tag{2.18}$$

Friction drag: From viscous shear stress at the wall
Pressure drag: Includes form drag (separation), induced drag (vortex system), and wave drag (negligible at low speed)

2.2.2.1 Induced Drag

Finite-span wings generate trailing vortices that induce downwash, tilting the lift vector and creating induced drag [6]:

$$C_{D,i} = \frac{C_L^2}{\pi e AR} \tag{2.19}$$

where AR is aspect ratio and e is span efficiency ($e \leq 1$, typically 0.7-0.9 for race wings). For low aspect ratio Formula Student wings ($AR \approx 2 - 3$), induced drag represents 30-50% of total drag.

2.2.3 Multi-Element Wing Aerodynamics

Formula Student front wings employ multiple elements to enhance downforce [6, 7]. Element interaction produces beneficial effects:

- **Slat effect**: Forward element reduces adverse pressure gradients on main element, delaying separation
- **Circulation effect**: Each element enhances total circulation
- **Dumping effect**: Wake from upstream elements convects away from downstream surfaces
- **Off-surface pressure recovery**: Pressure recovery in gaps/wake allows higher element loading

Gap and overlap between elements are critical design parameters [8].

2.2.4 Ground Effect

Ground effect describes enhanced aerodynamic performance for wings operating near a ground plane [9, 6] - fundamental to race car aerodynamics.

2.2.4.1 Physical Mechanisms

1. Venturi Effect: The gap between wing and ground forms a converging-diverging channel. Flow acceleration reduces static pressure beneath the wing per Bernoulli's equation:

$$p + \frac{1}{2}\rho u^2 = \text{constant} \tag{2.20}$$

2. Downwash Reduction: The ground acts as a symmetry plane, constraining the trailing vortex system. This reduces induced drag and effectively increases aspect ratio [6].
3. Boundary Layer Interaction: Ground boundary layer interacts with wing wake, affecting separation and pressure recovery.

2.2.4.2 End Plate Effects

End plates serve critical functions [7, 10]:

- Suppress tip vortices (prevent pressure leakage around wing tips)
- Increase effective aspect ratio
- Seal low-pressure region when extended to ground
- Generate strong controlled tip vortices

2.2.5 Flap Aerodynamics and DRS

2.2.5.1 Flap Deflection Effects

Trailing edge flap deflection alters effective camber and circulation [1]:

- **Lift increase**: $\Delta C_L \propto \delta_f \cdot (c_f/c)$ where c_f is flap chord
- **Moment change**: Nose-down pitching moment increases
- **Drag increase**: Both induced drag (higher C_L) and pressure drag (separation)

Flap effectiveness depends on: flap chord ratio, gap/overlap, Reynolds number, and deflection angle (effectiveness reduces at high δ_f due to separation).

2.2.5.2 Kutta Condition and Trailing Edge Flow

The Kutta condition requires smooth flow departure from sharp trailing edges with finite velocity [6]. This determines circulation strength and lift. When a flap deflects, the rear stagnation point moves, changing pressure distribution and circulation - the fundamental mechanism of DRS [8].

2.2.5.3 DRS Operation

For DRS applications, the flap typically deflects 10-30 degrees to reduce downforce and drag on straights, then returns to closed position for cornering. Understanding transient aerodynamics during this motion is the core objective of this thesis.

2.3 Vehicle Dynamics and Aerodynamic Forces

This section connects aerodynamic forces to vehicle performance, establishing the context for why transient DRS behavior matters for Formula Student vehicle dynamics.

2.3.1 Aerodynamic Force System

2.3.1.1 Force and Moment Definitions

Aerodynamic forces and moments acting on the vehicle are typically expressed in a vehicle-fixed coordinate system. For the front wing:
Downforce (negative lift): Vertical force pushing the vehicle toward the ground, increasing tire normal forces and thus cornering capability.
Drag: Horizontal force opposing motion, directly affecting acceleration and top speed.
Pitching moment: Moment about the lateral axis, affecting weight distribution and vehicle balance.
The moment coefficient:

$$C_M = \frac{M}{\frac{1}{2}\rho U_\infty^2 S_{ref} c_{ref}} \tag{2.21}$$

where c_{ref} is the reference length (wheelbase).

2.3.1.2 Aerodynamic Efficiency

Aerodynamic efficiency is measured by the lift-to-drag ratio:

$$\frac{L}{D} = \frac{C_L}{C_D} \tag{2.22}$$

For Formula Student downforce devices, high $|L/D|$ indicates efficient downforce generation with minimal drag penalty. Front wing $|L/D|$ typically ranges from 3-7 depending on configuration and ride height.

2.3.2 Influence on Vehicle Performance

2.3.2.1 Tire Force Generation

Aerodynamic downforce directly increases tire vertical load, enhancing lateral and longitudinal grip. The tire friction circle expands proportionally to normal force:

$$F_{tire,max} = \mu F_z = \mu(W + F_{aero,z}) \tag{2.23}$$

where μ is the tire friction coefficient, W is the static weight, and $F_{aero,z}$ is the aerodynamic downforce. For Formula Student vehicles, aerodynamic downforce can exceed vehicle weight at high speeds.

2.3.2.2 Cornering Performance

Maximum lateral acceleration in steady-state cornering is limited by available tire grip:

$$a_{y,max} = \mu g \left(1 + \frac{F_{aero,z}}{W}\right) \tag{2.24}$$

Front wing downforce contributes significantly to front tire load, affecting understeer/oversteer balance and maximum cornering speed.

2.3.2.3 Straight-Line Performance

Aerodynamic drag directly opposes acceleration. The drag force:

$$F_D = \frac{1}{2}\rho U^2 S_{ref} C_D \tag{2.25}$$

increases with the square of velocity, becoming the dominant resistance at high speeds. Peak power is consumed overcoming drag:

$$P_{drag} = F_D \cdot U = \frac{1}{2}\rho U^3 S_{ref} C_D \tag{2.26}$$

The cubic relationship with velocity explains why drag reduction (via DRS) is most beneficial at high speeds.

3

Methods

This chapter describes the computational methodology employed for the transient analysis of the active aerodynamics system on the Formula Student front wing. The study encompasses geometry development, mesh generation, solver configuration, and post-processing procedures.

3.1 Overview of Simulation Strategy

The computational investigation follows a systematic approach:

1. **Geometry preparation**: CAD model development of front wing with DRS flap mechanism
2. **Mesh Preparation**: Mesh using overset mesh to accompany moving geometries
3. **RANS Initialization**: Simulation initialization at various vehicle speeds
4. **Transient DES simulations**: Time-accurate analysis of DRS actuation cycles
5. **Parametric study**: Evaluation of different flap angular velocities convergence studies

3.2 Geometry and CAD Modeling

3.2.1 Front Wing Design

The front wing is a slightly modified front wing from CFS26 and consists of multiple elements. The main element (main segment) has the biggest chord and the span covers the width of the car, with the flap element having a smaller chord and also covering less than half of the span of the main segment. The wing is designed to operate at a nominal ride height of 50 mm above the ground plane, which is a safe standard for scrape prevention during pitch. At the edge of the main segment there are endplates that help with sealing the underflow and creating controlled vortices for the wheel wake management

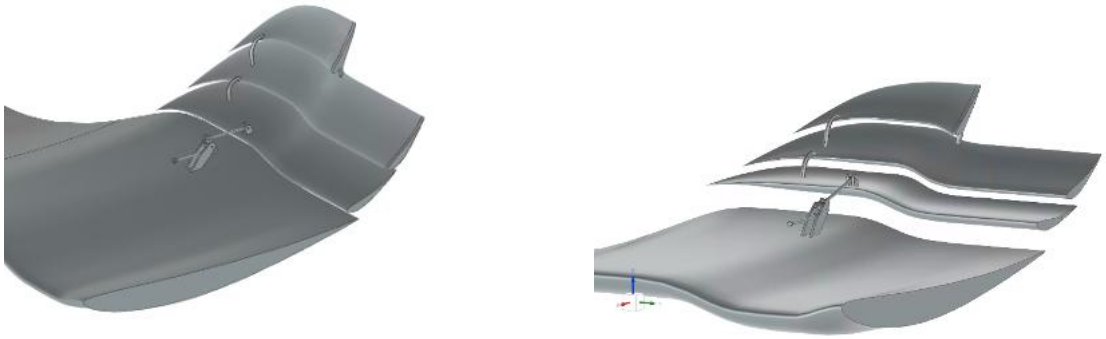
Figure 3.1: Front Wing CAD Assembly.

3.2.2 DRS Flap Mechanism

The DRS (Drag Reduction System) includes all of the main flaps. The flap rotates about a pivot point positioned at 62% chord of the first flap element. The maximum deflection angle is 20 degrees, and all other flaps follow the movement of the first flap as can be seen in Figure 3.2.
The flap motion is prescribed as a time-dependent rotation:

$$\theta(t) = \begin{cases} 0 & t < t_{start} \\ \omega \cdot (t - t_{start}) & t_{start} \leq t \leq t_{end} \\ \theta_{max} & t > t_{end} \end{cases} \tag{3.1}$$

where ω is the angular velocity (deg/s) and θ_{max} is the maximum flap deflection angle.



(a) DRS closed **(b)** DRS open

Figure 3.2: Front wing CAD Model in Closed and Open DRS Configurations.

3.2.2.1 Actuation

The actuation mechanism uses a simple pivot mechanism to transform the linear movement to the angular movement of the flaps. To start with the first flap has a mounting point for a rod. That rod connects to the rocker that is mounted to the main segment and is designed to take the linear motion of the actuator rod and transform it into an arc motion around the pivot point of the flap, which is the attachment point to the endplates. As the actuator is not in the scope of this thesis, but only the movement is, the rod is left floating, but there should be a linear actuator mounted to the main segment as annotated in Figure 3.3.

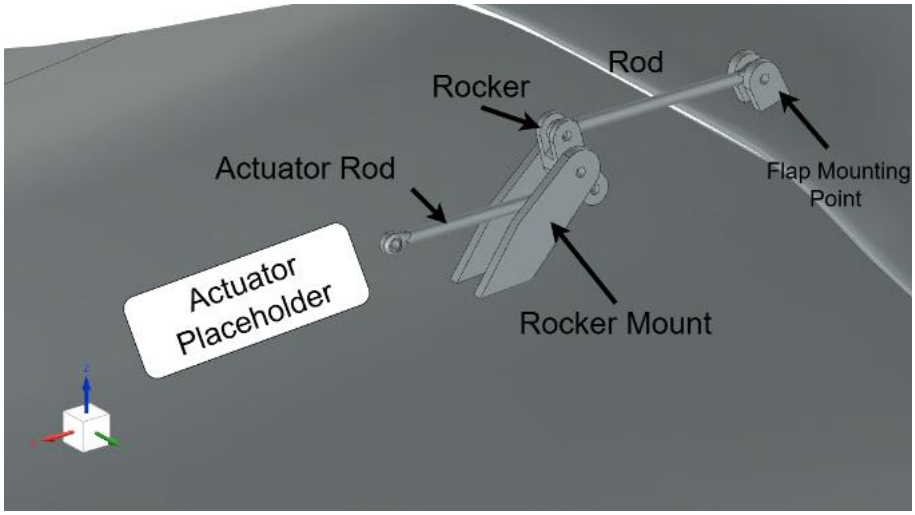
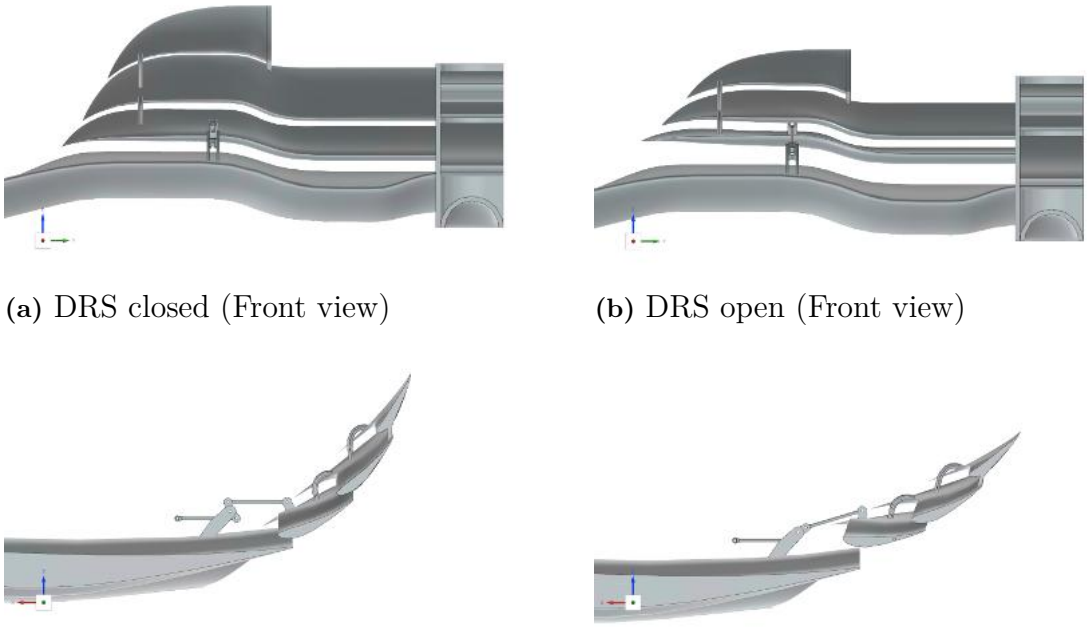


Figure 3.3: Actuation Mechanism.

To verify that the movement is adequate, in the CAD software the movement of the actuation can be simulated by applying a linear motion in the X direction (red arrow) and evaluate the response of the flaps, based on the rockers geometry design the flaps should open up to 20 degrees as can be seen in Figure 3.3.



(a) DRS closed (Front view)

(b) DRS open (Front view)

(c) DRS closed (Side view)

(d) DRS open (Side view)

Figure 3.4: Front Wing Mechanism CAD Movement.

From Figure 3.4 it can be seen that the extension of the actuator rod leads the flaps to rotation with its center being the pivot point on the endplate.

3.2.2.2 Actuation Cover

To reduce the effects of the mechanics to the flow a cover is designed for the moving parts of the actuation mechanism. The cover also helps with simplifying the model, as the actuation parts now are placed in a flow separation area they can be removed. This simplification helps with reducing the cell count needed and also the movement simplification, if simplifications are not made the rod that connects to the flap will need a new motion with X,Z translation and Y-Moment specifications.

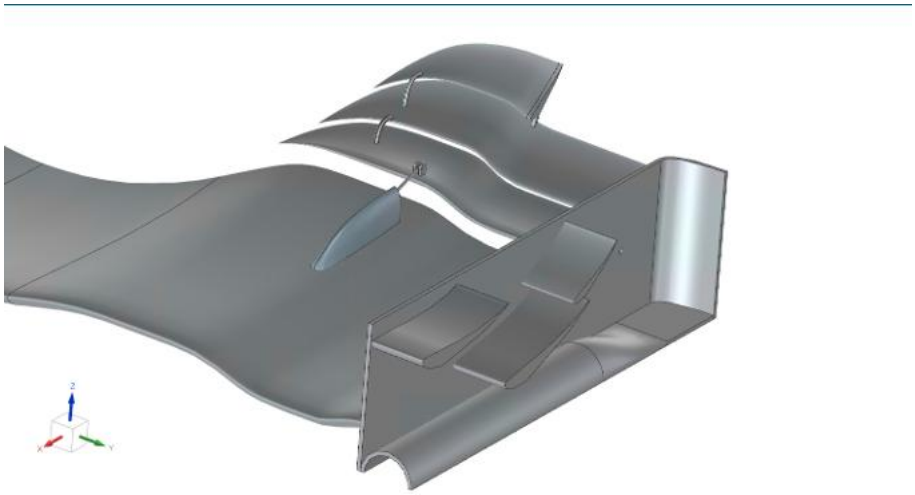


Figure 3.5: Actuation Cover.

With the actuator cover, the result is the Figure 3.6 below in the CFD simulation software.

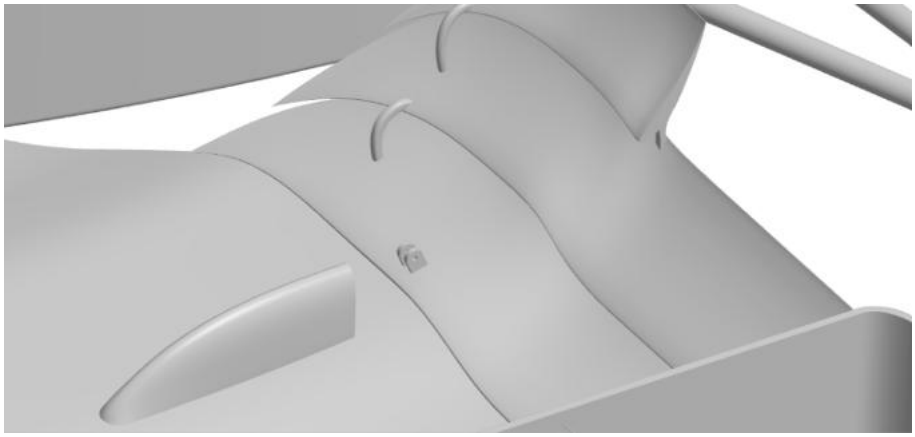


Figure 3.6: Actuation Mechanism in Simulation Software (STAR-CCM+).

3.2.3 Complete CAD Model

The final CAD Geometry used for the simulations is the complete car developed for CFS26 with the modified Front wing.

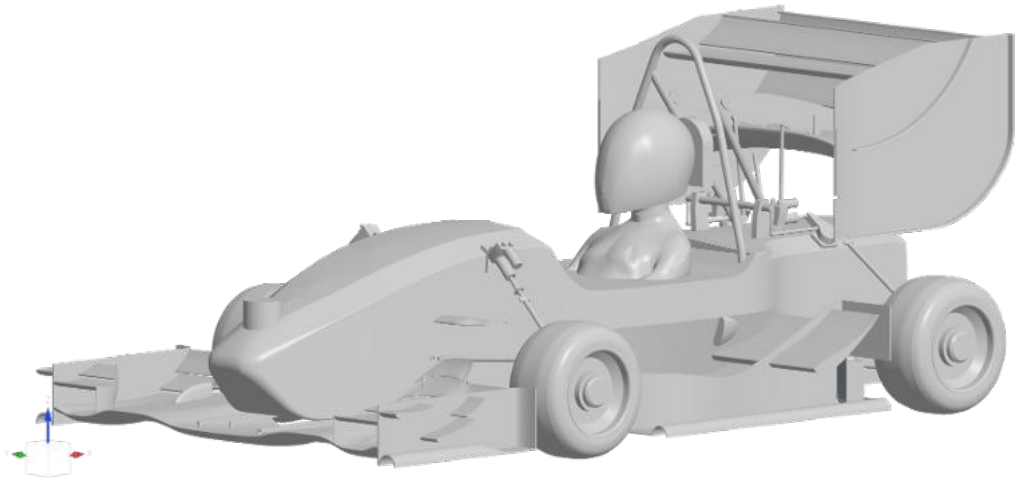


Figure 3.7: Geometry used in CFD Simulation Software (STAR-CCM+).

Although the full car simulation is more computationally expensive and complex than the front wing alone, the effect of the front wing as the first aerodynamic device transfers throughout the car and affects all parts of the car. Mostly other aerodynamic surfaces like the rear wing and the side wing as well as the wheels which are rotating. A core component of the thesis is to evaluate how the other parts are affected by the movement of the front wing flaps.

3.2.4 Computational Domain

The computational domain extends sufficiently to minimize blockage effects and ensure proper wake development:

- Upstream: Four car lengths
- Downstream: Seven car lengths
- Lateral: Ten car widths
- Height: Eight car heights

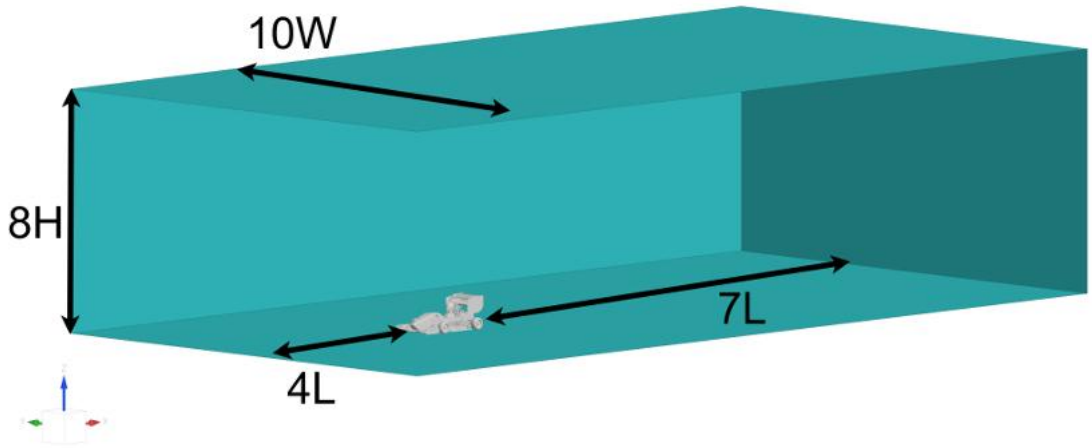


Figure 3.8: Wind Tunnel Used for CFD Simulations.

The inlet is set up as a velocity inlet with the speed set based on the current simulation. The outlet is a Pressure Outlet with the pressure being the Reference pressure (Atmospheric) The ground plane is modeled as a moving wall with velocity equal to the freestream velocity to simulate the relative motion between the vehicle and ground and prevent boundary layer forming as well as rotating wheels. The wind tunnel ceiling and side walls are set up as symmetry planes.

3.3 Mesh Generation

The computational mesh needs to meet multiple criteria, some of them are low y^+, stability while rotating geometries, good wake refinement. In the following section it is presented how the desired mesh was achieved.

3.3.1 Meshing Strategy

To start with Trimmed cell mesh is used as its typically used for vehicle aerodynamics due to the directed main flow. This approach allow for the mesh to be fairly consistent between modifications.[11]

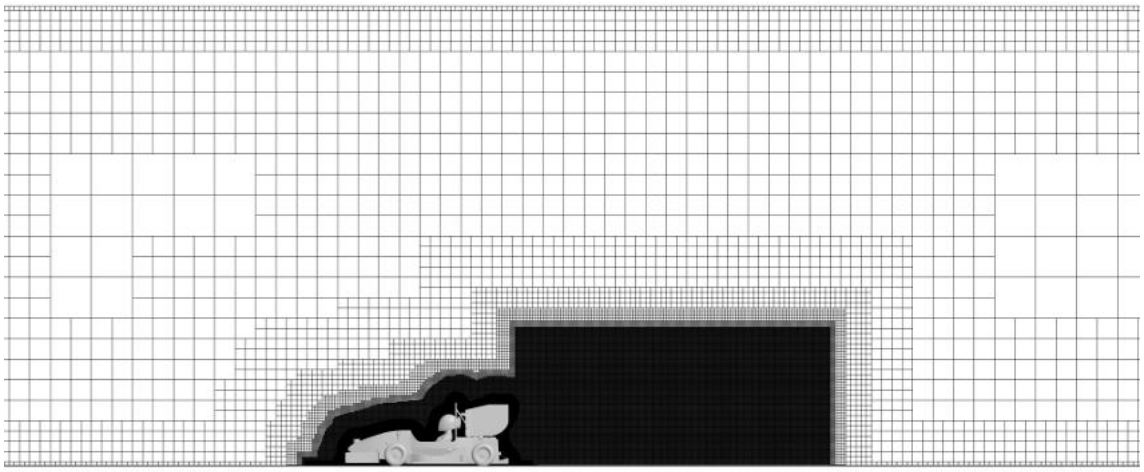


Figure 3.9: Trimmed Cell Mesh.

For the refinement Geometry offsets where used for a more efficient near car refinement and then for the wake refinement rectangular volumes where used as can be seen in 3.9. For the car refinement two offsets where used using the car geometry, 0.2 m and 0.5 m where sufficient for good transition and good near car accuracy.

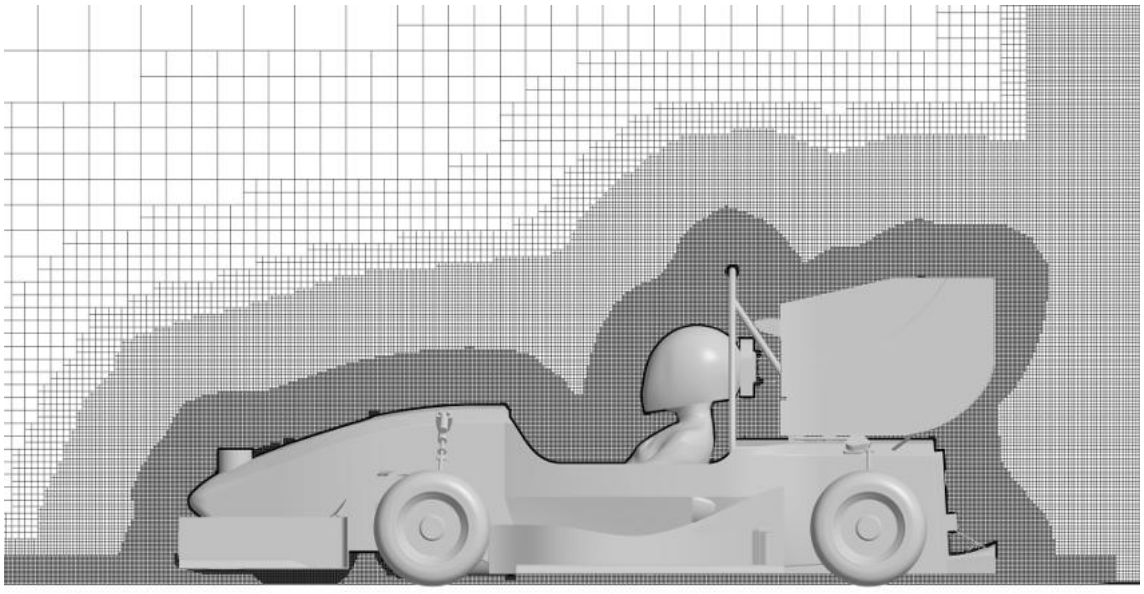


Figure 3.10: Mesh Refinement Offsets.

The values used for refinement are in the Table 3.1. There is also a flap overset which would be analyzed and explained in the sub section 3.3.3 as it is mainly used to accommodate the moving geometry.

Table 3.1: Mesh Size Distribution.

Base size	18 mm
Offset 0.5 m	12 mm
Offset 0.2 m	6 mm
Flap Offset	3 mm

3.3.2 Boundary Layer Resolution

For accurate RANS results near solid surfaces, the boundary layer is resolved with prism layers in this scenario a low y^+ is needed. The target for this is $y^+ < 3$. To achieve that the following prism layer setup was used.

Table 3.2: Prism Layer Setup.

First cell height	6.255×10^{-5} m
Number of prism layers	12
Growth ratio	1.5
Total prism layer thickness	8 mm

For the front wing custom total prism layer thickness has been set to 3mm as it allow for more cells in the gap between front wing flaps and main segment which is needed to have acceptable donor cells between the two geometries.

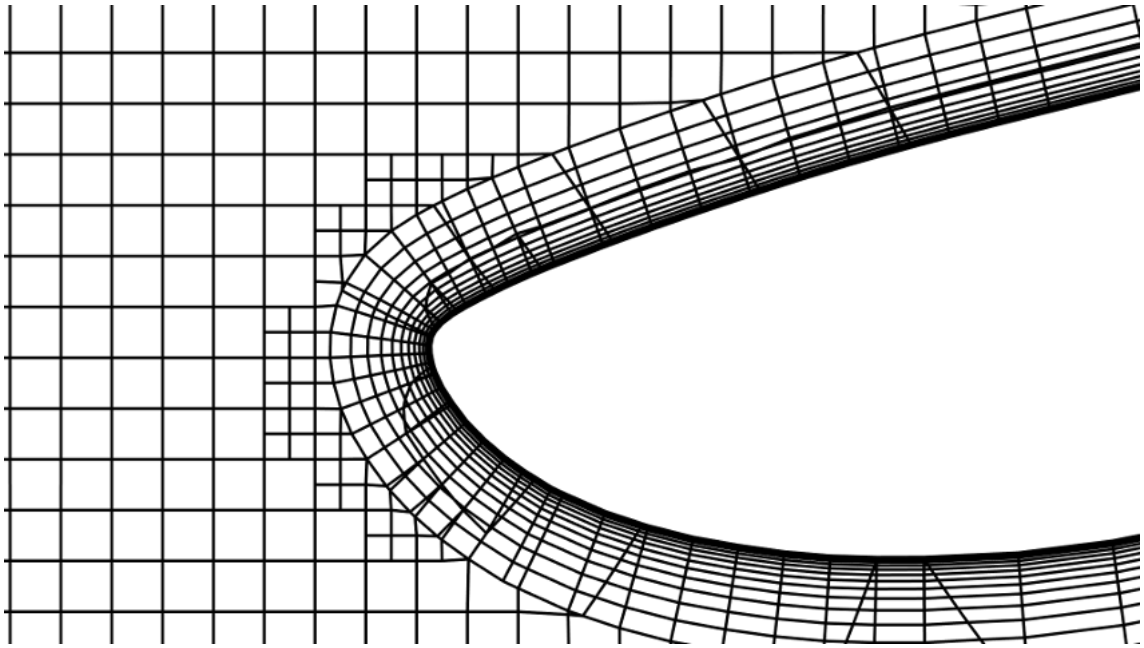
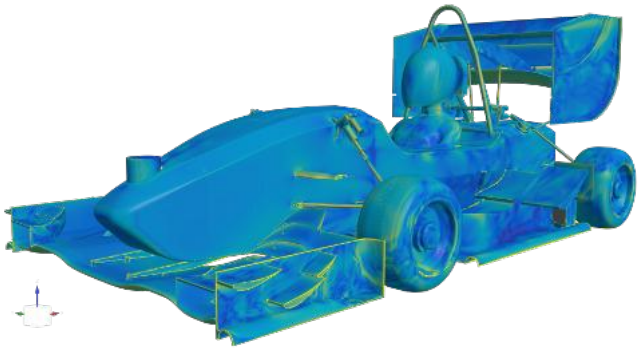


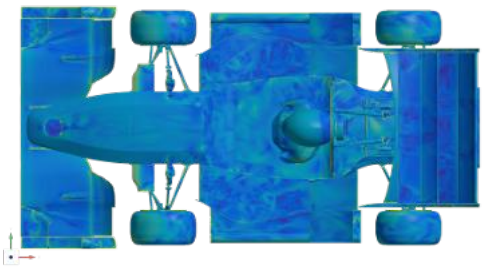
Figure 3.11: Front Wing First Flap Leading Edge Prism Layers.

The y^+ on the car surface can also be visualized. The target of $y^+ < 3$ its not achieved fully with some cells being closer to $y^+ = 5$. This values capture the near-wall physics reasonably well. Achieving $y^+ < 3$ across the entire car's complex

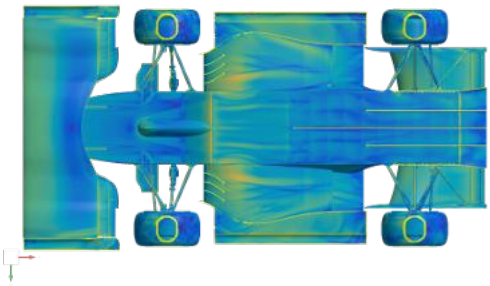
geometry would require significant mesh refinement. The trade off between cell count and wall resolution is acceptable in this case since again most of the surface maintains a good y^+ value.



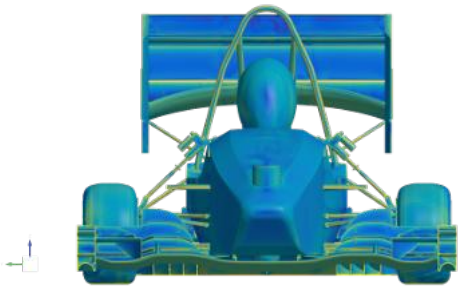
(a) y^+ Isometric



(b) y^+ Top



(c) y^+ Bottom



(d) y^+ Front

(e) y^+ Back

Figure 3.12: y^+ Scalar on Car Surface.

3.3.3 Moving Mesh Strategy for DRS Actuation

For the transient DRS actuation simulations, an Overset mesh has been used, also known as Chimera or overlapping meshes, is an advanced meshing strategy in computational fluid dynamics (CFD) designed to handle complex geometries with significant mesh motion. The Overset mesh involves creating multiple mesh regions that overlap and move independently. The key components are:

- **Background Mesh**: A stationary base mesh
- **Overset Region**: One or more moving/deforming mesh regions that can be superimposed on the background mesh

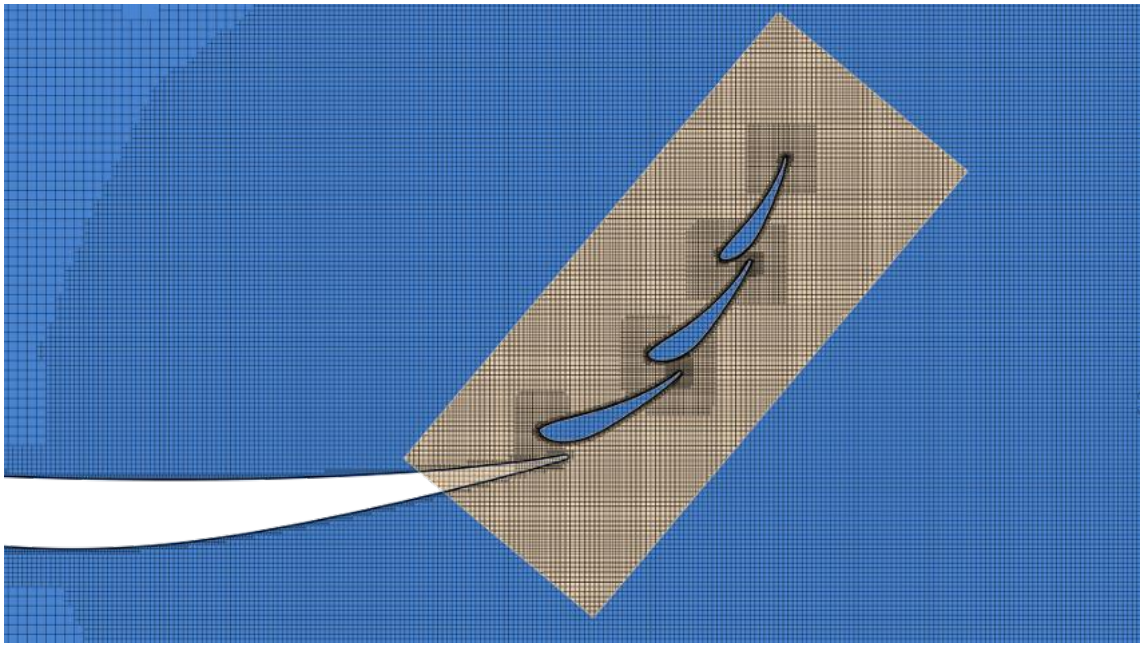


Figure 3.13: Background Mesh and Overset Region at $y = 0.48m$.

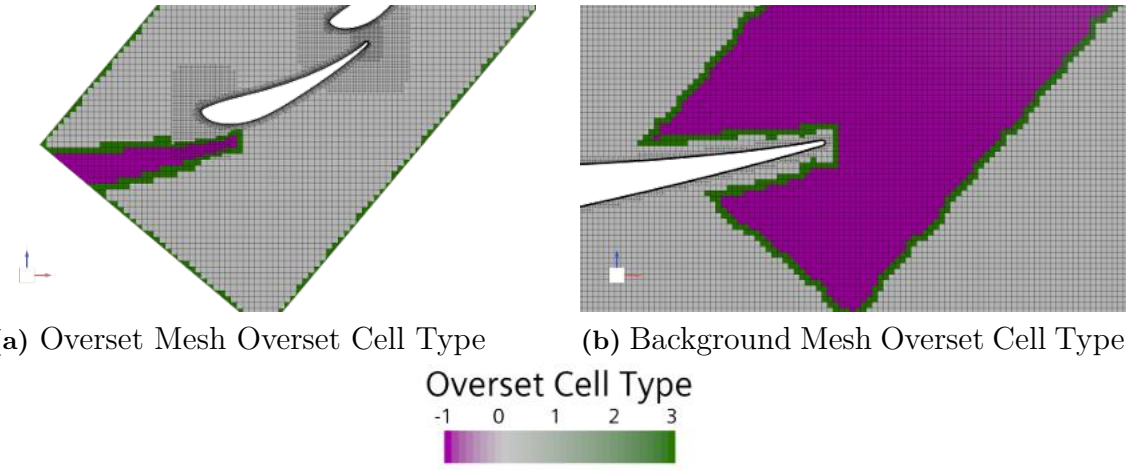
As can be seen in Figure 3.13 the Overset Mesh (Brown) overlaps the Background Mesh (Blue). The background mesh is the main mesh that is used for the rest of the car as can be seen by the Front wing main segment but the flaps are blue because they are removed from the Overset Mesh region. It can also be noted that the Front wing main segment is not removed from the Overset Mesh, that will be done during the Mesh Assembly Process 3.3.3.1.
An important rule when using overset mesh is that the overset and background mesh cells should be of similar size in the overlap region. This is done by using an offset refinement of the flaps as seen in Table 3.1 and then having a strict cell size in the overset volume mesh.

3.3.3.1 Mesh Assembly Process

The mesh starts with Hole-Cutting which it is essentially the method which overlapping meshes are integrated and communicate flow-field information. It also determines which cells are:

- **Active**: Cells where the equations are solved.
- **Inactive**: Cells where the equations are not solved.
- **Acceptor cells**: Cells that transfer information between Overset and Background Meshes

To further understand how Overset Mesh works the scalar of "Overset Cell Type" can be checked, this indicates Active cells ($Overset\ Cell\ Type = 0$), Inactive cells ($Overset\ Cell\ Type = -1$), Acceptor cells ($Overset\ Cell\ Type = 3$). This can be seen in the following Figures 3.14



(a) Overset Mesh Overset Cell Type **(b)** Background Mesh Overset Cell Type

Figure 3.14: Overset Cell Types.

From Figure 3.14, it can be extracted that the inactive cells of one mesh are active cells for the other while the active and inactive cells have a "border" of acceptor cells between them.

The second critical step for the Mesh Assembly Process is the Donor Search, which ensures that each acceptor cell in one mesh has corresponding donor cells in another mesh. In the image below an example of acceptor and donor cells is shown, two acceptor cells are shown using dashed lines, one in the background mesh and one in the overset mesh

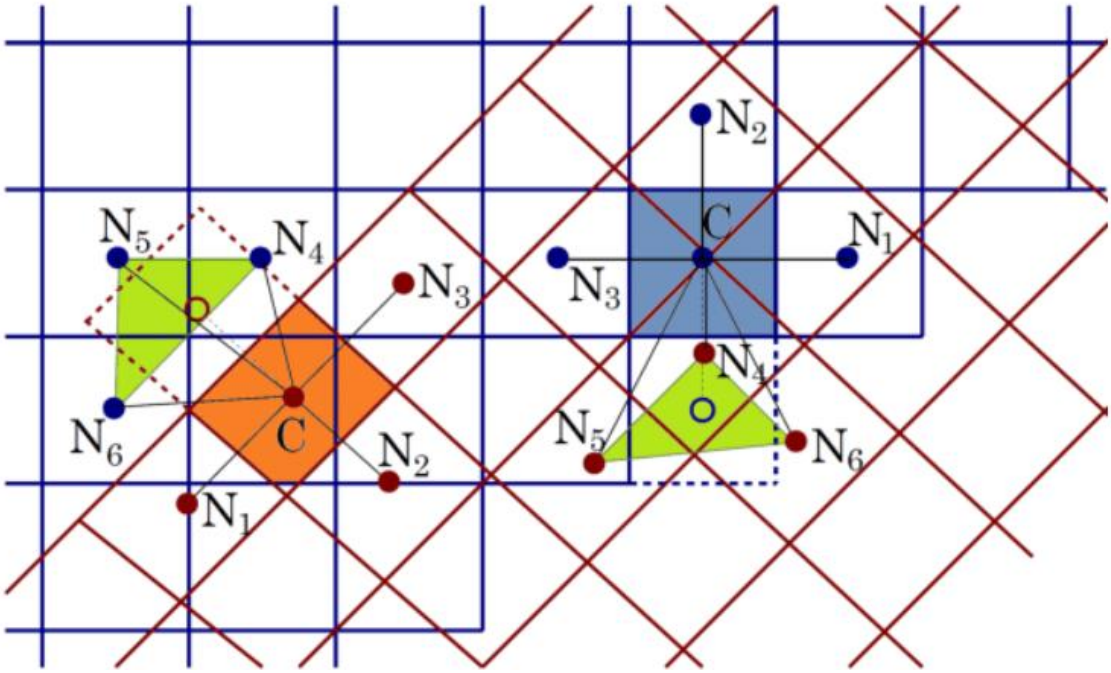


Figure 3.15: Example of Acceptor and Donor Cells.

The fluxes through the cell face between the last active cell and the acceptor cell are approximated in the same way as between two active cells. However, whenever the variable value at the acceptor cell centroid (marked by the open circles in the above figure) is referenced, the weighted variable values at the donor cells are substituted:

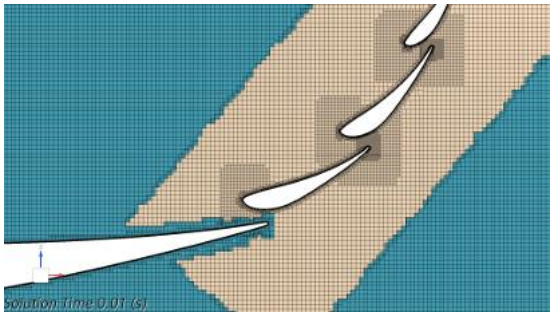
$$\Phi_{acceptor} = \sum \alpha_i \Phi_i \tag{3.2}$$

Where α_i are the interpolated weighting factors, Φ_i are the values of the dependent variables Φ at donor cells N_i.

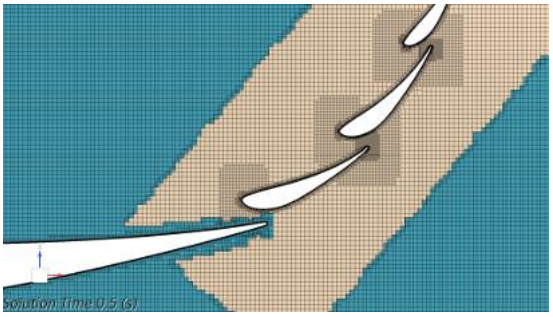
Where the donor and acceptor cells interact there is an overset interface where the different meshes interact to exchange flow field information.

3.3.3.2 Mesh Movement

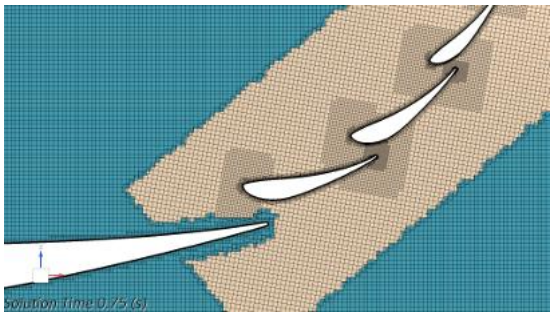
The flap rotation is prescribed according to Equation 3.1, and the Overset Mesh Region moves while the flaps stay "stationary" within it.



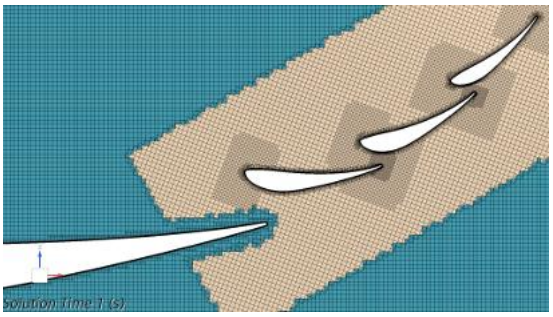
(a) Mesh Interaction $t = 0s$



(b) Mesh Interaction $t = 0.5s$



(c) Mesh Interaction $t = 0.75s$

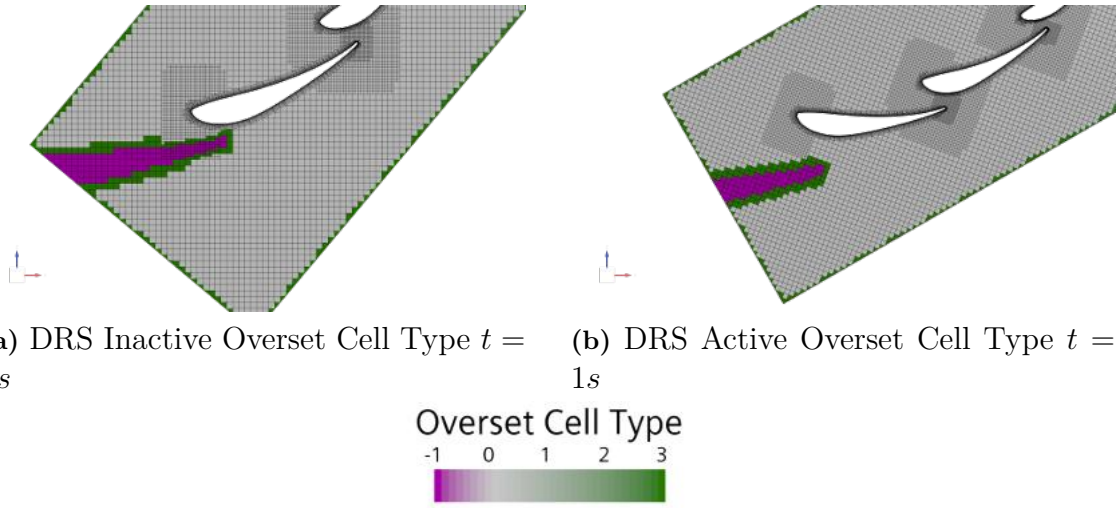


(d) Mesh Interaction $t = 1s$

Figure 3.16: Mesh and Geometry Movement - XZ Plane.

The above Figures 3.16 shows the rotation of the overset region. Based on the Equation 3.1 the values are $t_{start} = 0.5s$, $\theta_{max} = 20°$ based on the limitations of the actuation mechanism, and $\omega = 40°\text{s}^{-1}$. Based on the values above it can be seen that the movement takes $0.5s$ to transform the flaps to the expected angle of $\Delta\theta = 20°$.

Based on the last Section 3.3.3.1 the interface between the two meshes needs to be updated after every movement. This means inactive cells may become active or acceptor cells and vice versa. This can be seen in the Figure 3.17 as its evident that the inactive cells are fewer and not at the same place in the two time steps.



(a) DRS Inactive Overset Cell Type $t = 0s$

(b) DRS Active Overset Cell Type $t = 1s$

Figure 3.17: Overset Cell Type Change Between Active and Inactive DRS.

3.4 CFD Solver Configuration

All simulations are performed using STAR-CCM+ version 2602. Two simulation approaches are employed: steady-state Reynolds-Averaged Navier-Stokes (RANS) for flow initialization and computation time reduction. While Detached Eddy Simulation (DES) captures the transient flow dynamics as well as the effect of the moving geometries to the flow field and force generation.

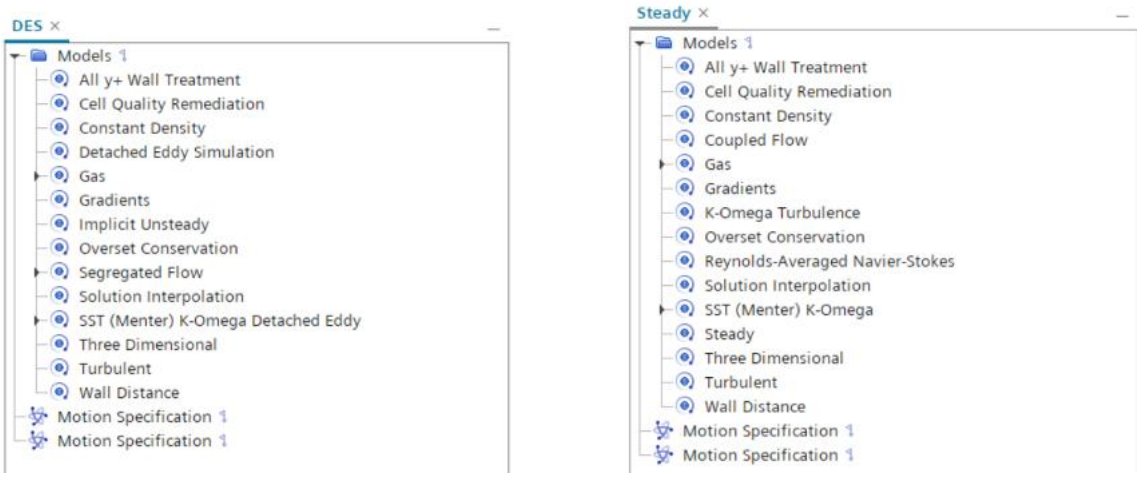
3.4.1 Boundary Conditions

The following boundary conditions are applied consistently across both RANS and DES simulations:

- **Inlet**: Velocity inlet with U_∞ = Depending on scenario m/s
- **Outlet**: Pressure outlet with $p_{gauge} = 0$ Pa
- **Ground**: Moving wall with velocity $U_{ground} = U_\infty$
- **Vehicle surfaces**: No-slip wall with zero velocity
- **Top and sides**: Symmetry boundary condition

3.4.2 Stage Configuration

To start with how the simulation is set up so it uses stages, Stages are typically used for complex scenarios and also to establish a steady state field with RANS then transitioning to an unsteady state with rigid body movement, in this case DES. The simulation is set up with four stages, all four stages are not needed but its easier to automate the simulation this way.



(a) DES Stages **(b)** RANS Stage

Figure 3.18: Stages Setup.

All DES Stages have the same Physics Model while the Steady Stage is set up with RANS. Another difference is the Motion Specification of the Overset Regions, For DES it is ω, DES_RM which is short for Reversed Motions the motion is specified as $-\omega$ and finally for Steady and DES_NM which stands for No Motion the Motion Specification is set up as Stationary.

3.4.3 Turbulence Model Configuration

Both simulation approaches utilize variants of the SST (Menter) k-ω turbulence model as can be seen in Figure 3.18, which is well-suited for external aerodynamics with its ability to handle adverse pressure gradients and flow separation. The IDDES formulation provides shielding of boundary layers from grid-induced separation while enabling LES-like resolution of separated regions, capturing large-scale unsteady structures in the wake. All parameters follow SIEMENS recommendation [11] and can be found in Appendix A. The simulations run for sufficient physical time to allow transient flow structures to develop and achieve statistical convergence of time-averaged aerodynamic quantities. For the setup Siemens's guide manual for external aerodynamics was used. [11]

3.5 Simulation Cases and Test Matrix

3.5.1 Steady-State RANS Initialization

Steady-state RANS simulations establish a good initialization point for the transient simulation.

3.5.2 Transient DES Phases

The transient DES simulations capture the complete DRS actuation cycle. Each simulation consists of five distinct phases:

1. **Phase 1 - Flush baseline**: DRS closed ($\theta = 0°$), flow reaches statistical steady state
2. **Phase 2 - Opening flaps**: Flap rotates from 0° to θ_{max} at angular velocity ω
3. **Phase 3 - Hold open**: Flap held at θ_{max}, flow re-stabilizes
4. **Phase 4 - Closing flaps**: Flap rotates from θ_{max} back to 0° at angular velocity ω
5. **Phase 5 - Hold closed**: Return to baseline configuration

Multiple angular velocities are tested to understand the influence of actuation rate on transient aerodynamic behavior.
All car speeds and angular velocities are shown in Table 3.3

3.5.3 Test Matrix

The Test Matrix is a matrix that has basic information for each simulation scenario that that was run.

Table 3.3: Test matrix for transient DES simulations.

Simulation ID	U_∞ $\left[\frac{km}{h}\right]$	ω $\left[\frac{degrees}{s}\right]$	Δ_θ $[°]$
DRS_20_60	60	20	20
DRS_20_80	80	20	20
DRS_40_60	60	40	20
DRS_40_80	80	40	20
DRS_80_80	80	80	20

3.5.4 Time Step Selection

Time step selection for the transient DES simulations follows siemens recommendations [12] which are based on the smallest cell size in the overlapping zone . The time step is selected such that the flap rotates approximately $\Delta\theta = 0.04°$ per time step as it fulfills the needs of the overset mesh:

$$\Delta t = \frac{\Delta\theta}{\omega} \tag{3.3}$$

This results in the following time steps for each actuation rate:

Table 3.4: Time Step Selection for Each Simulation Case.

Simulation ID	Δt $[s]$
DRS_20_60	0.002
DRS_20_80	0.002
DRS_40_60	0.001
DRS_40_80	0.001
DRS_80_80	0.0005

This approach ensures smooth flap motion while maintaining computational efficiency, with each 20° rotation requiring approximately 500 time steps regardless of actuation rate.

3.6 Post-Processing and Data Analysis

3.6.1 Force Coefficient Calculation

Aerodynamic forces are non-dimensionalized using the vehicles frontal area A and freestream dynamic pressure.

$$C_L = \frac{L}{\frac{1}{2}\rho U_\infty^2 A}, \quad C_D = \frac{D}{\frac{1}{2}\rho U_\infty^2 A} \tag{3.4}$$

where A is the Frontal area of the car, U_∞ is the freestream velocity, ρ the density of the fluid, L the calculated lift and D the calculated drag.

3.6.2 Aerodynamic Balance

As mentioned in Section 2.3 aerodynamic balance is very important for motorsports application to have a stable and predictable car around the track.

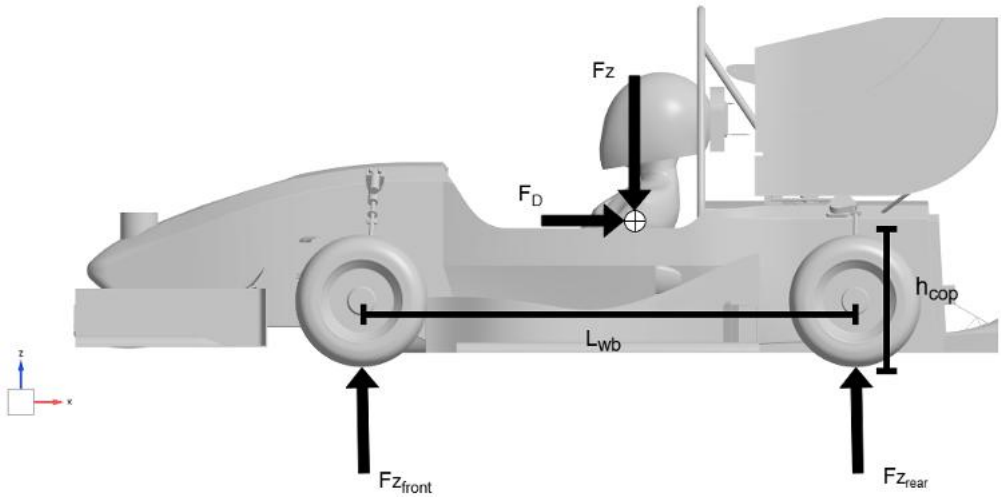


Figure 3.19: Balance Free Body Diagram.

$$\sum M_{rear} = 0: \quad F_{z,front} \cdot L_{WB} + F_D \cdot h_{cp} - F_{z,total} \cdot b = 0 \tag{3.5}$$

$$F_{z,front} = \frac{F_{z,total} \cdot b - F_D \cdot h_{cp}}{L_{WB}} \tag{3.6}$$

$$F_{z,rear} = F_{z,total} - F_{z,front} \tag{3.7}$$

where b is the distance from the rear axle to the center of pressure, L_{WB} is the vehicle wheelbase, $F_{z,total}$ is the total aerodynamic downforce, F_D is the aerodynamic drag force, and h_{cp} is the height of the center of pressure above the ground plane.
Equations 3.5 - 3.7 describe how the balance of the vehicle is calculated. The Center of Pressure (CoP) is found using the simulation software.

3.6.3 Transient Force History Analysis

For each DRS actuation cycle, the following metrics are extracted:

- C_L and C_D for all time steps
- L and D per part of the car
- Aerodynamic Balance

This metrics are used to compare angular velocities as well as freestream velocity differences during the transient simulations. Normalized values will be used throughout the report as flow trends have a greater significance than absolute magnitudes for comparative analysis.

3.6.4 Flow Field Visualization

Flow field visualization is very important for aerodynamic analysis to extract the flow behavior at different angles of the DRS operation; this analysis includes:

- Pressure coefficient C_p distributions on vehicle surfaces

- Vorticity magnitude contours in wake regions
- Q-criterion iso-surfaces for vortex identification
- Visualization of the front wing flaps - flow interaction using streamlines

3.7 Summary

This chapter presented the DRS mechanism used for the actuation as well as the geometry used for the simulations which is simplified to save cells for small parts in wake regions. The Boundary conditions of the wind tunnel used where introduced as well as the models used for the simulation for both RANS and DES solver.
The specialized overset mesh is explained and how an adequate time step is chosen, there is no mesh dependency study as the thesis follows best practices used from the CFS team which are established over the years
The simulation setup and cases that are going to be implemented has been introduced as well as basic parameters for each simulation setup. Finally, the post processing parameters and scalar fields have been presented as well as how each one contributes to the overall objective of the thesis.

4

Results

This chapter presents the computational results from the transient DRS aerodynamics study. The analysis is organized in three parts: detailed flow field analysis of a representative case, transient force response comparison across different actuation rates, and overall drag reduction performance across all operating conditions.

4.1 Simulation Matrix

Table 4.1 summarizes the completed simulations. Two vehicle speeds (60, 80 km/h) are analyzed with DRS angular velocities of 20, 40, and 80 deg/s.

Table 4.1: Completed Simulation Matrix Showing Vehicle Speed and DRS Angular Velocity Combinations.

Vehicle Speed (km/h)	**Angular Velocity ω (deg/s)**		
	20	40	80
60	✓	✓	-
80	✓	✓	✓

The case with vehicle speed of 60 km/h (16.67 m/s) and $\omega = 40$ deg/s is selected for detailed flow analysis as it represents intermediate conditions and exhibits representative transient behavior.

4.2 Detailed Flow Analysis

This section presents a comprehensive analysis of the flow field evolution during the complete DRS actuation cycle for the selected case ($V = 60$ km/h, $\omega = 40$ deg/s). The flow field will be analyzed at two distinct time stamps.

- Start of phase 3 (End of DRS activation)
- End of phase 5 (Final time step)

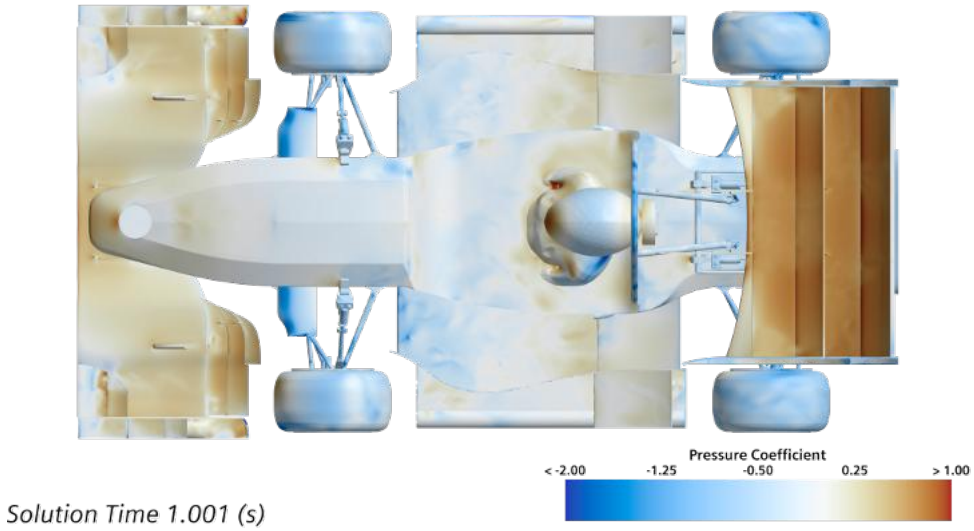
For each time stamp, important vortex structures and flow characteristics will be analyzed.

4.2.1 Pressure Distribution

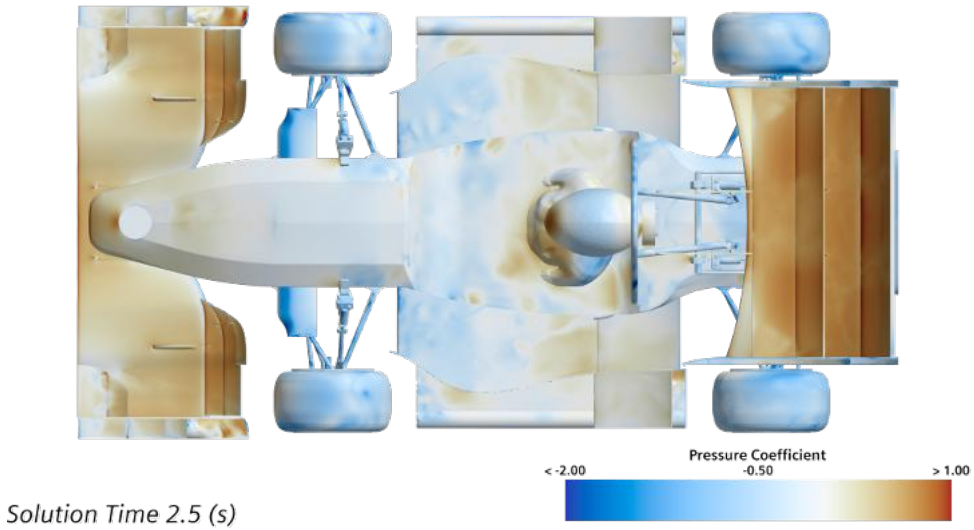
In this section, the pressure distribution will be analyzed on the car surface, focusing on important parts of the aeropackage and how DRS affects them.

4.2.1.1 Top View

The analysis starts with the Overall distribution at the top of the car. For the analysis originally three Phases where considered but the difference between Phase 1 (end of flush) and Phase 5 weren't large enough for the comparison to be compared. Therefor the analysis continues with a two phase comparison between Phase 3 and Phase 5.



(a) Pressure Distribution Phase 3 Top View

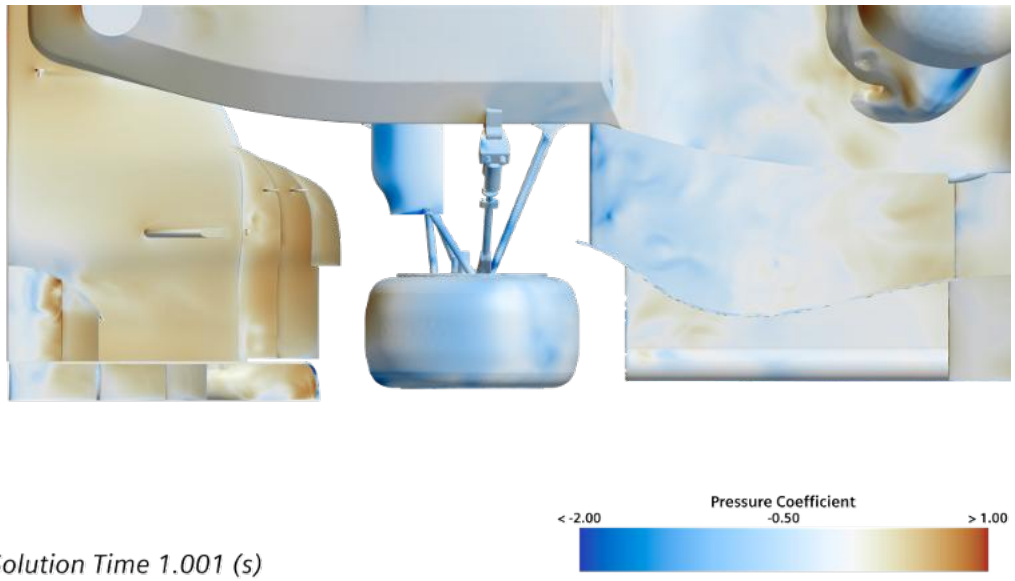


(b) Pressure Distribution Phase 5 Top View

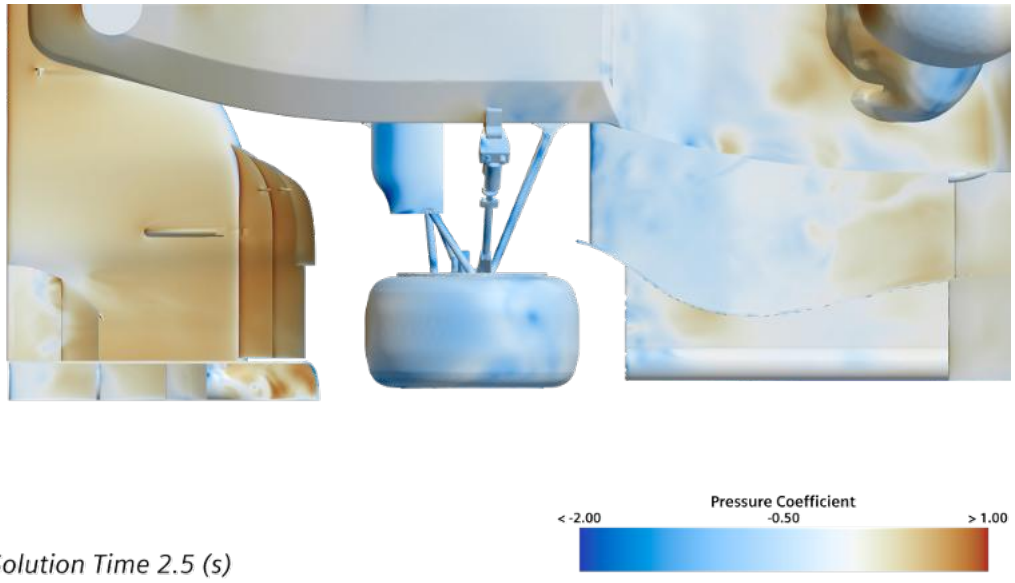
Figure 4.1: Pressure Distribution Top View.

From the top view a big difference can be seen on the pressure on the front wing between active and inactive DRS which is expected with the lower angle of attack. Between Phase 1 and Phase 5 not enough of a difference can be seen in the pressure field, the field has been investigated further in the postprocessing software. As discussed before only Phase 3 and 5 will be compared as Phase 1 and 5 are very

similar so Phase 1 will not be included for the rest of the Flow Field Analysis Section. From the top view interesting interactions can be seen between the side wing as well as the down washing elements, So for the next comparison the focus is on that area.



(a) Pressure Distribution Phase 3 Side Wing



(b) Pressure Distribution Phase 5 Side Wing

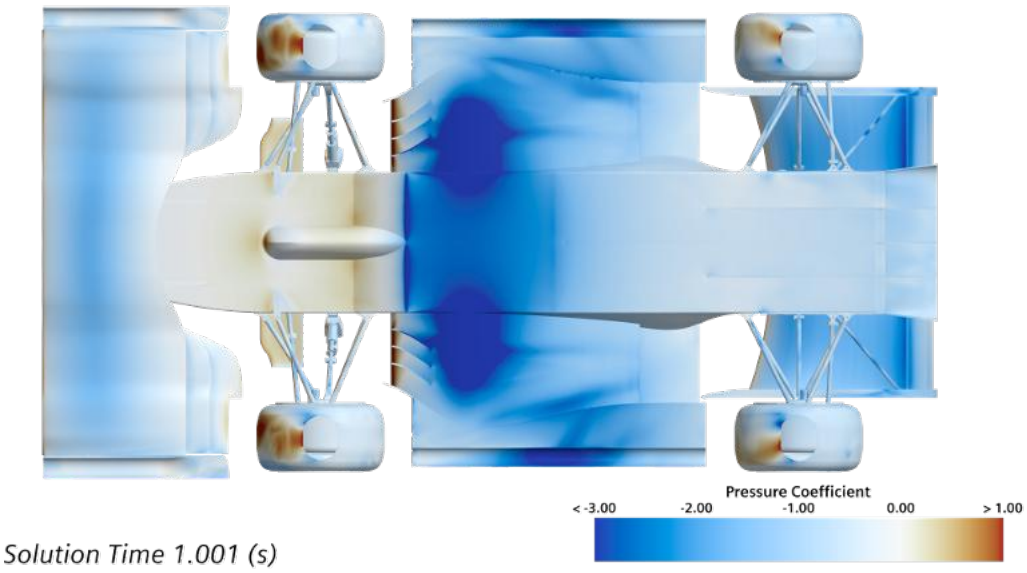
Figure 4.2: Pressure Distribution Side Wing.

Starting with the front wing the most evident difference is the pressure distribution on the front wing flaps. It can also be seen that the pressure in the main segment of the front wing is affected by this and the pressure near the flaps is not as high. Moving to the wheels, lower pressure can be seen on top of the wheel during Phase 5 this induces lift which is undesirable.
The down washing elements difference becomes more evident in Figure 4.4 where the elements under side can be seen, from the top side some difference is evidence

but not enough to come to a conclusion.
Finally, for the side wing the focus is on the leading edge of the main segment to start with, during Phase 3 the low pressure detached area is smaller then during Phase 5. Closer to the trailing edge of the side wing main segment a larger high pressure area can be seen. For the top side of the side wing floor leading edge it can be seen that during Phase 5 the wheel wake is not managed as well as in Phase 3, resulting in a low pressure area, this can also be seen in streamline Figure **??**.

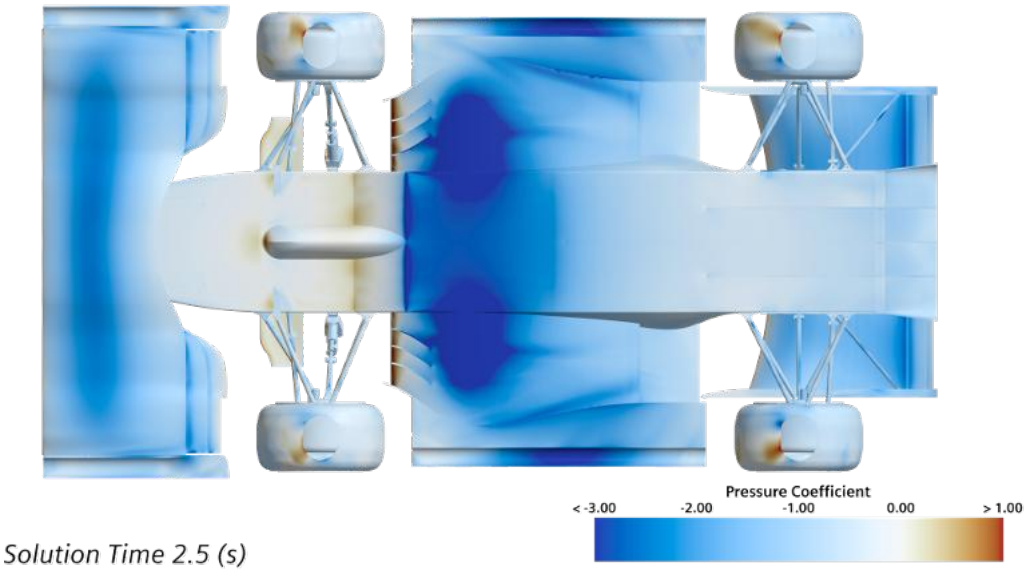
4.2.1.2 Underbody

As the top of the car has been analyzed, the analysis now moves to the underbody to evaluate if there are any differences between the two Phases



(a) Pressure Distribution Phase 3 Underbody

Figure 4.3: Pressure Distribution Underbody.



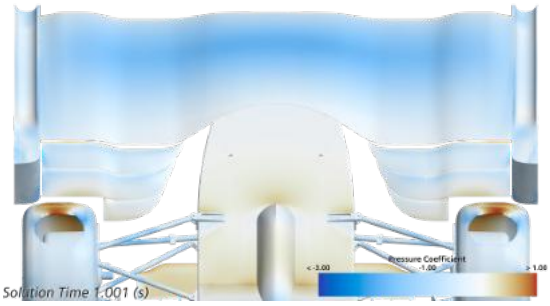
(b) Pressure Distribution Phase 5 Underbody

Figure 4.3: Pressure Distribution Underbody (continued).

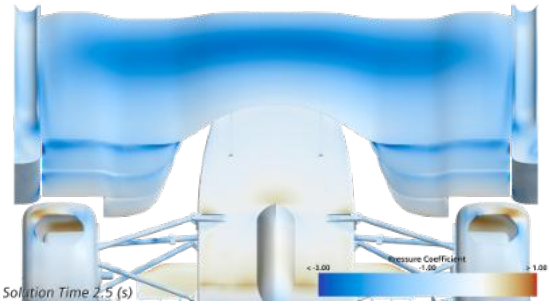
From Figure 4.3, considerable pressure distribution differences can be seen. The analysis will start from the front of the car, examining each component in depth.

4.2.1.2.1 Front Wing

Starting with the front wing, some observations about the down washing elements will be made as well.



(a) Pressure Distribution Phase 3 Front Wing Underside



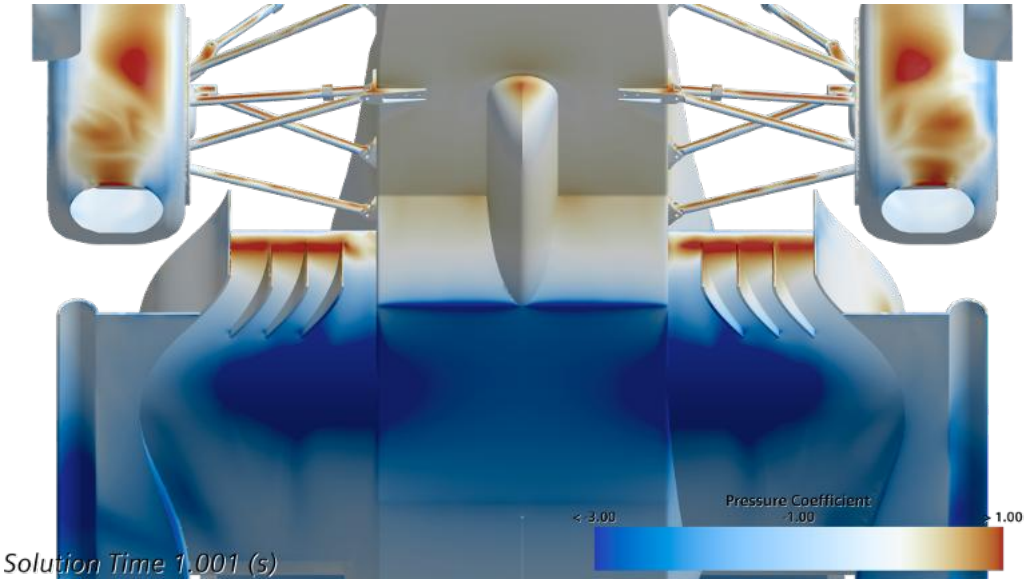
(b) Pressure Distribution Phase 5 Front Wing Underside

Figure 4.4: Pressure Distribution Front Wing Underside.

Starting with the main segment, a larger low pressure area can clearly be seen throughout the whole surface of the element that continues to all the flaps.
Continuing to the down washing element,a higher pressure is observed during Phase 3. The final observation is the high pressure area on the contact patch of the wheel, this will be touched upon during Section 4.3.2.

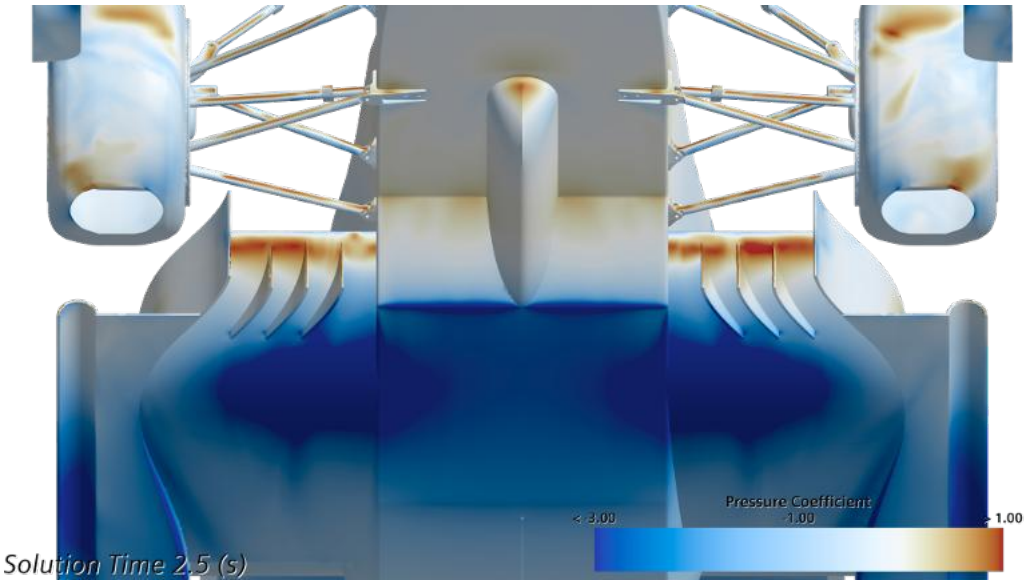
4.2.1.2.2 Side Wing

Moving on to the side wing.



(a) Pressure Distribution Phase 3 Side Wing Underside

Figure 4.5: Pressure Distribution Side Wing Underside.



(b) Pressure Distribution Phase 5 Side Wing Underside

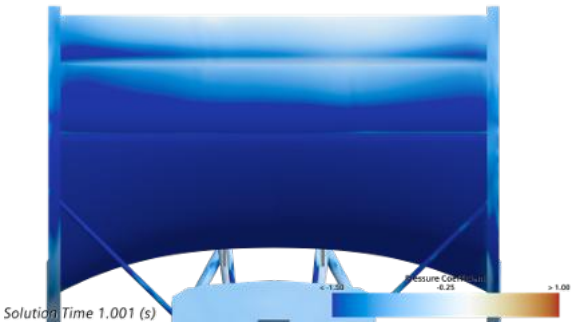
Figure 4.5: Pressure Distribution Side Wing Underside (continued).

As presented in Figure 4.5, the low pressure distribution on the monocoque as well as on the side wing is greater during Phase 5 then Phase 3 leading to greater ground effect while DRS is not active.
Another difference is the two stagnation areas on the leading edge of the main segment of the side wing as well as the wheel as mentioned in Paragraph 4.2.1.2.1.

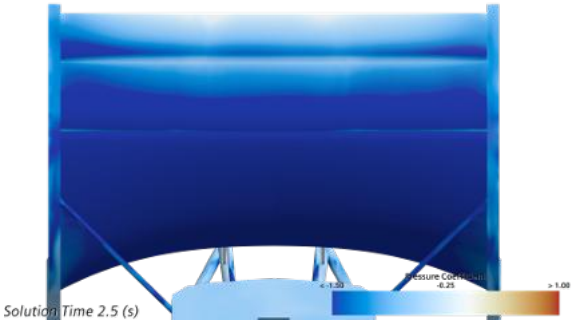
During Phase 3 the two stagnation areas are larger which limit the effectiveness of the DRS system.

4.2.1.2.3 Rear Wing

Finishing off with the Rear wing, the effects of the DRS will be seen further downwind.



(a) Pressure Distribution Phase 3 Rear Wing Underside



(b) Pressure Distribution Phase 5 Rear Wing Underside

Figure 4.6: Pressure Distribution Rear Wing Underside.

A larger low pressure area can be identified during Phase 5 on both the second and third elements of the rear wing which is expected as the rear wing is optimized for flow without the DRS, further analysis of the rear wing force generation will be made during the Section 4.3.2.

4.2.2 Vortex Formation and Shedding

To make the vortices more apparent, the focus will be on Q-criterion and the vorticity magnitude on planes. The Q-criterion is a vortex identification method distinguishing regions where rotation dominates over strain in the flow. Defined as:

$$Q = \frac{1}{2}(||\Omega||^2 - ||S||^2) \tag{4.1}$$

where Ω represents the rotation rate tensor and S represents the strain rate tensor.

Q-Criterion Isosurface Isometric view will be used to locate strong vortices then they will be analyzed based on vorticity magnitude on an Y-Z Plane as the out-washing/in-washing vortices are the ones of most interest.

First , the focus will be on Phase 5 with the changes of Phase 3 being highlighted.

4.2.2.1 Vortex Formation Analysis

For Phase 5 the analysis starts with a macroscopic view of the Q-criterion Isometric view.

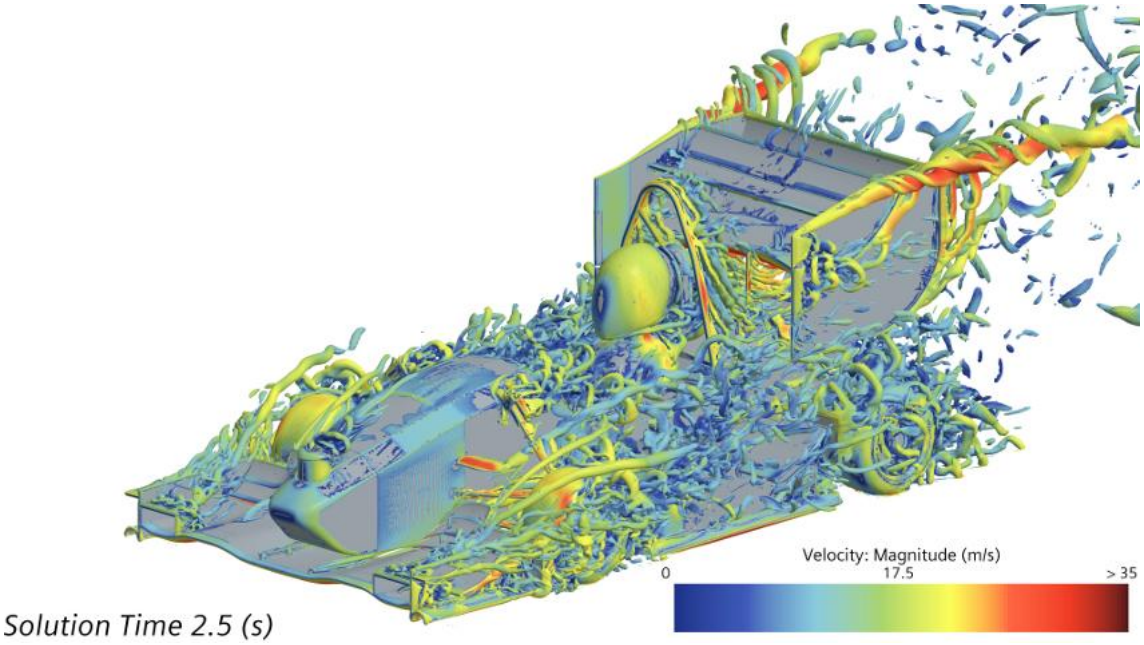


Figure 4.7: Velocity Scalar Q-Criterion Isosurface Isometric View of Phase 5.

Figure 4.7 shows Q-criterion iso-surfaces identifying coherent vortical structures during Phase 5.
The six vortices that are observed taking into account only half car are:

- Front Wing Endplate
- Front Wing Out-washing Element
- Front Wing Flaps Tip
- Front Wing Flaps Outer
- Down-washing Element tip
- Rear Wing Endplate

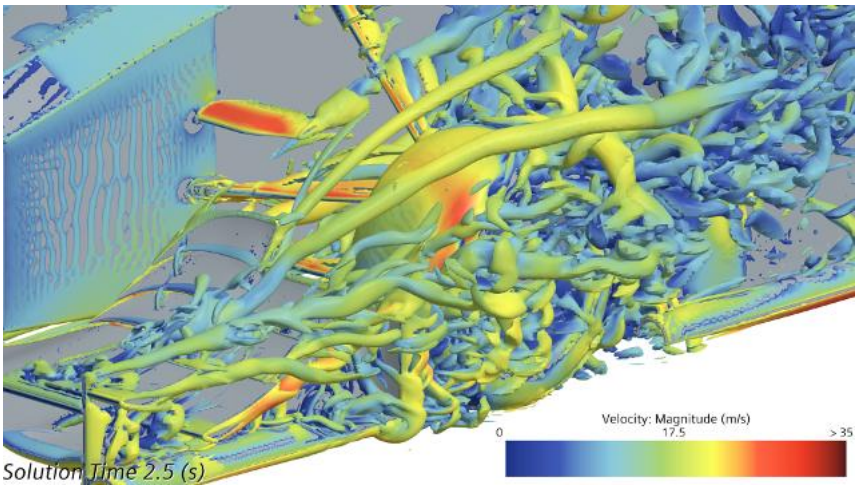


Figure 4.8: Velocity Scalar Q-Criterion Isosurface Isometric View of Phase 5 Wheel Region.

Using the above scalar visualizations seen in Figure 4.8 and an additional vorticity magnitude scalar field as shown in Figure 4.9 vortex structures can be followed and analyzed based on how they break down and how the flow field is affected.

4.2.2.1.1 Out Washing Element and Front Wing Endplate

The first vortex identified is the out-washing element vortex. The vortex becomes evident as the Front Flaps leak around the vertical element. In the following Figure 4.9 the red line can be seen pointing the path of the vortex and the Black arrow annotating the rotational direction of the vortex as the scalar that is used is the magnitude rotational direction does not show in the scalar field.

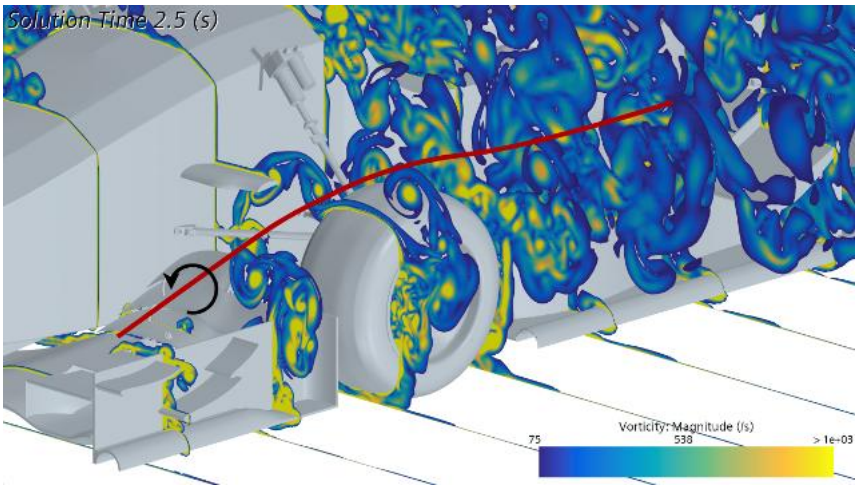


Figure 4.9: Vorticity Magnitude Out Washing Element Annotated Vortex Isometric View.

The vortex leaves the front wing region while out and up-washing. The vortex interacts with the endplate vortex creating a contra rotating vortex couple which

one enhances the other. Following both vortices it is possible to see where each one breaks down. The Endplate vortex helps with outwash on the outside of the tire and then breaks down just after. While the out washing element vortex moves over the wheel, confining the wake and out washing it. It breaks down midway through the side wing main segment. For Phase 3 the biggest change is that the contra rotating vortices still have the same interaction but are closer to each other and are on the top edge of the wheel. This helps keep the out washing element vortex go further than Phase 5 as it keeps itself energized using the endplate vortex.

4.2.2.1.2 Front Wing Flaps and Down Washing Element

Following the other 3 vortices mentioned in Subsection 4.2.2, the analysis can start with the Down washing element vortex as it is more evident in the isometric view, for the other vortex structures they will need to be analyzed through two dimensional plane sections.

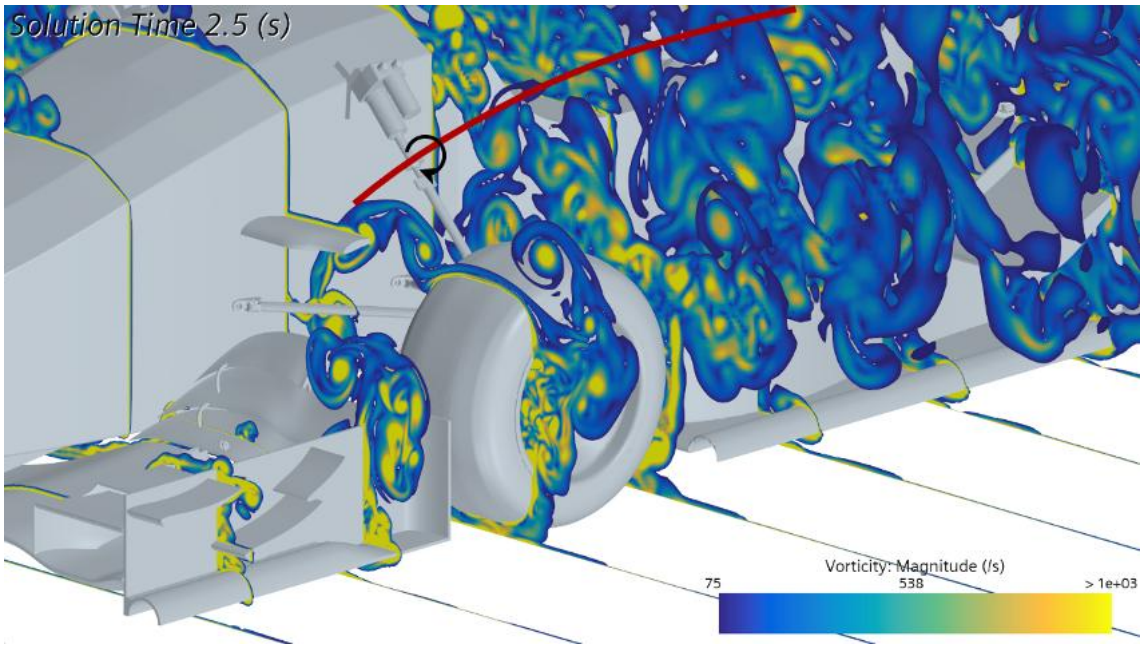
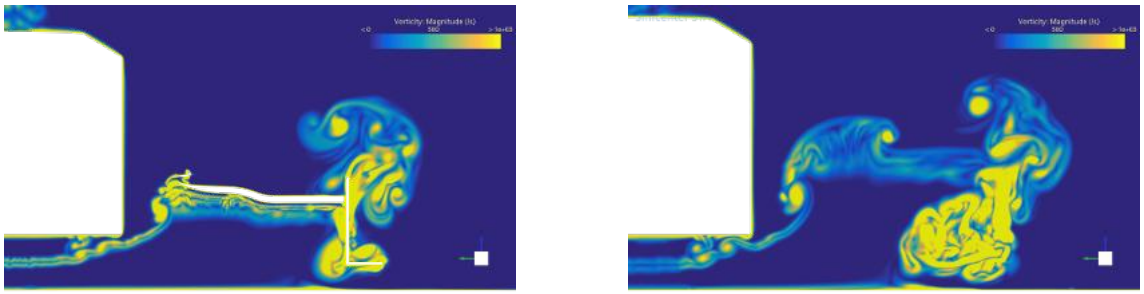


Figure 4.10: Vorticity Magnitude Down Washing Element Annotated Vortex Isometric View.

The down washing element creates strong interaction with the suspension wake as well as "seal" the wake from the cockpit.

The two other vortices form at the flap tips and on the third flap mounting edge at $x = 0.75m$ and $x = 0.85m$ respectively.



(a) Front Wing Flap Tip Vortex Formation - Vorticity Magnitude Plane $x = 0.75m$.

(b) Front Wing Third Flap Vortex Formation - Vorticity Magnitude Plane $x = 0.85m$.

Figure 4.11: Front Wing Flap Vortex Formation.

Continuing with the the flap vortex interactions, the wing tip vortex interacts with the higher suspension wishbone removing energy from the vortex. The vortex of the third flap moves to the top left of the wheel and manages the wheel wake by out-washing. To conclude, the vortices are followed while interacting again with the suspension dampers as well as the wishbones before breaking down. The down washing element continues to seal the cockpit area until it dissipates near the driver. For Phase 3 the down washing element vortex seems to have the same effects as Phase 5.The flap vortices are very different as the tip vortex doesn't form and the third flap vortex is lower then before not managing the wheel wake as well as when the DRS is inactive.

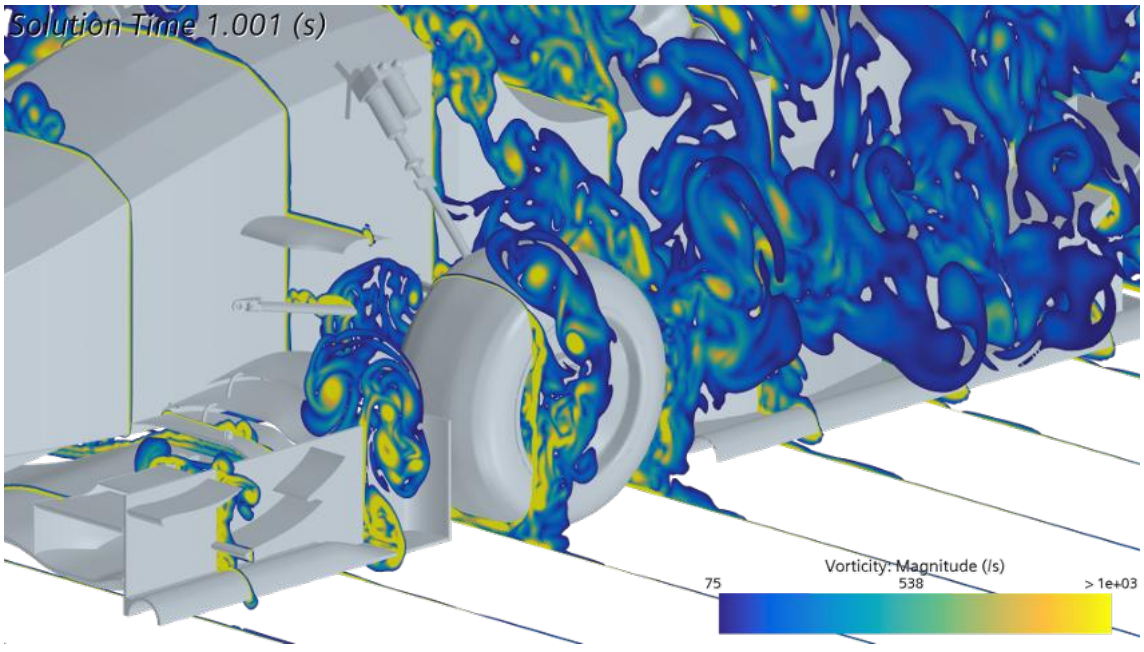


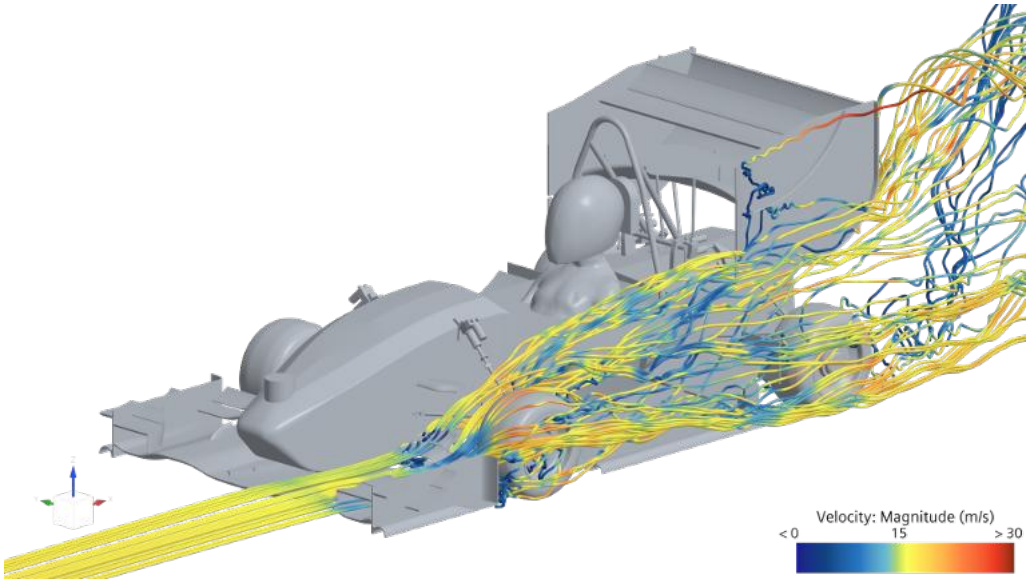
Figure 4.12: Vorticity Magnitude Wheel Region Isometric View - Phase 5.

4.2.2.2 Vortex Formation Summary

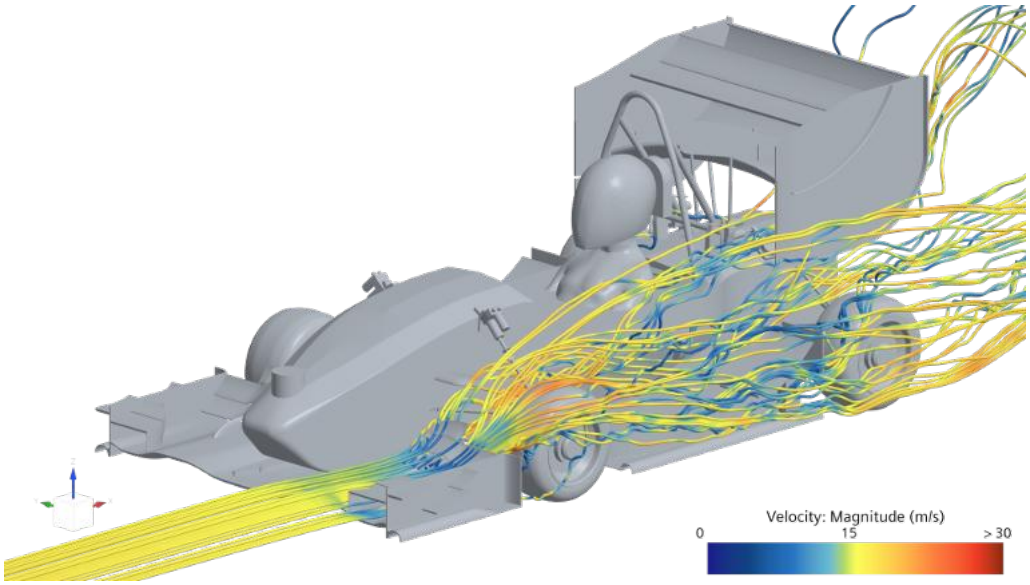
In summary, the vortices remain largely consistent across configurations, with only minor differences. The exception is the front wing flap tip vortex, which does not form when the DRS is open. With the DRS active the pressure difference across the flap tip drops, and without a high enough pressure gradient. the tip vortex fails to develop.

4.2.3 Flow Comparison Based on Flap Angle of Attack

To analyze how the flow reacts to the different angle of attack of the flaps streamlines that have the flaps as the seed part where used. This creates streamlines across the span and chord of the Flaps.



(a) Streamlines Isometric View Open Flaps



(b) Streamlines Isometric View Closed Flaps

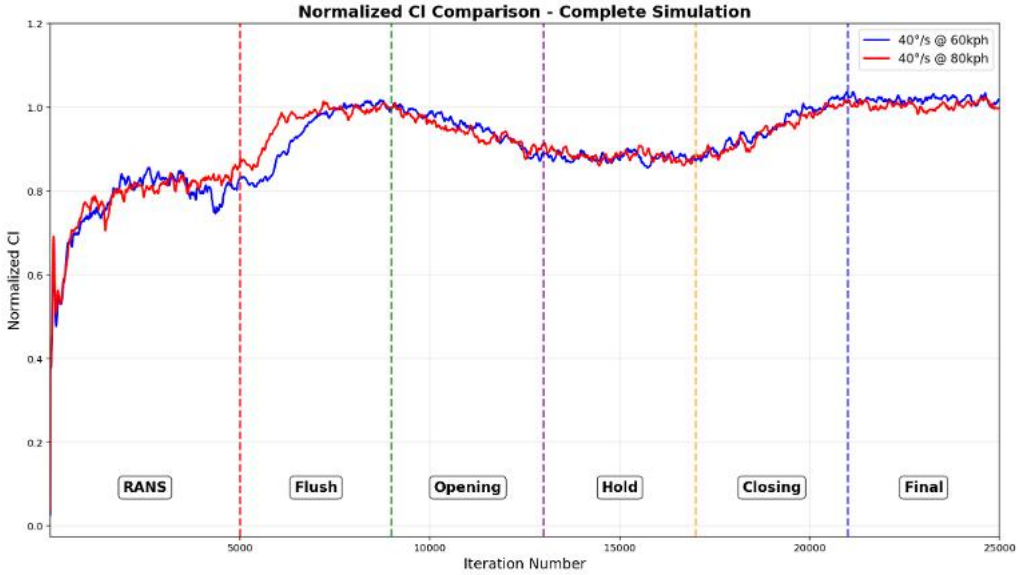
Figure 4.13: Streamlines Isometric View.

From Figures 4.13 the analysis can start with the flow coming from the front wing flaps. By focusing on the endplate - front wing flaps - wheel interaction in Figures 4.13, it can be observed that while DRS is inactive most of the flow from the outer section of the flaps goes over the wheel in contrast to the flow while the DRS is active the flow goes around the wheel. The same interaction can be seen with the

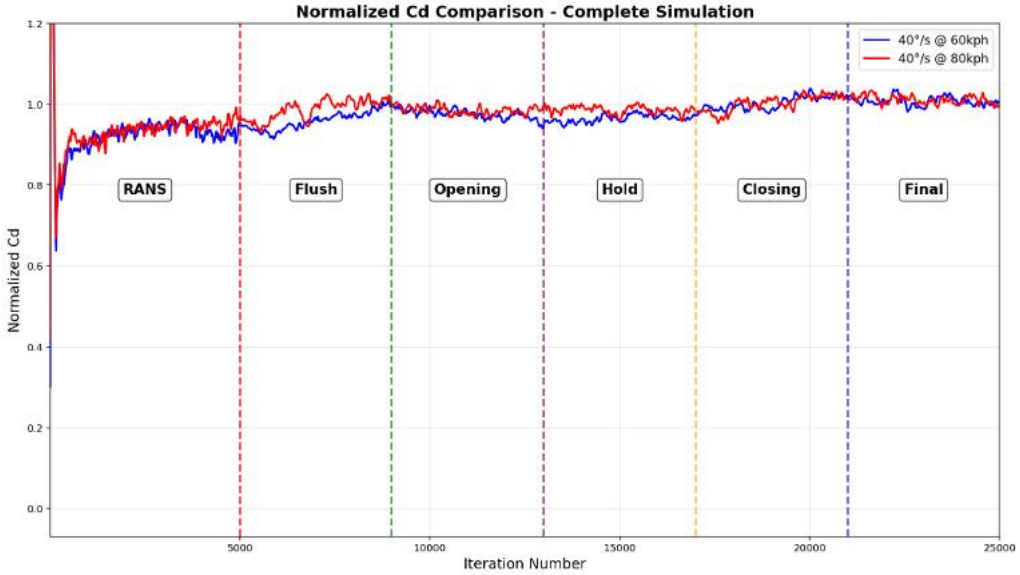
down washing elements which with DRS active the flow doesn't interact with the geometry but for DRS inactive the flow splits between the pressure and suction side of the down-washing element. It is also evident that the amount of up wash that the flaps produce while DRS is active is not as pronounced as in the inactive configuration, the down washing element is fully covered from the flow of the flaps reaching almost up to the spring and damper assembly.

4.3 Aerodynamic Forces Data Postprocessing

To analyze the forces that are being generated from the aerodynamic package, monitors are exported from the simulation software and import them into a Python code that has been written to export important plots and also filter data due to their noisy nature. The scripts that have been used are given in Appendix B. For this Section the simulations DRS_40_60 and DRS_40_80 will be used.
subsectionForce Coefficients
To start with, the Normalized force coefficients for the complete simulation are plotted.



(a) Normalized Cl for Complete Simulation



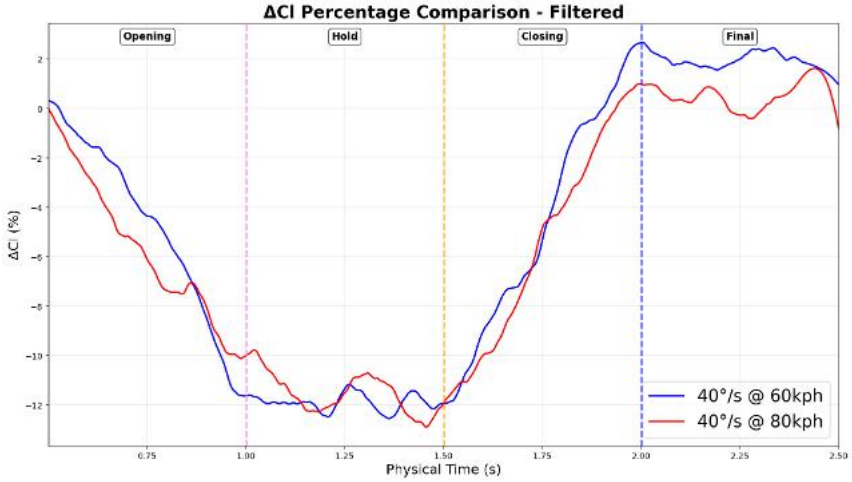
(b) Normalized Cd for Complete Simulation

Figure 4.14: Normalized Force Coefficient Comparison.

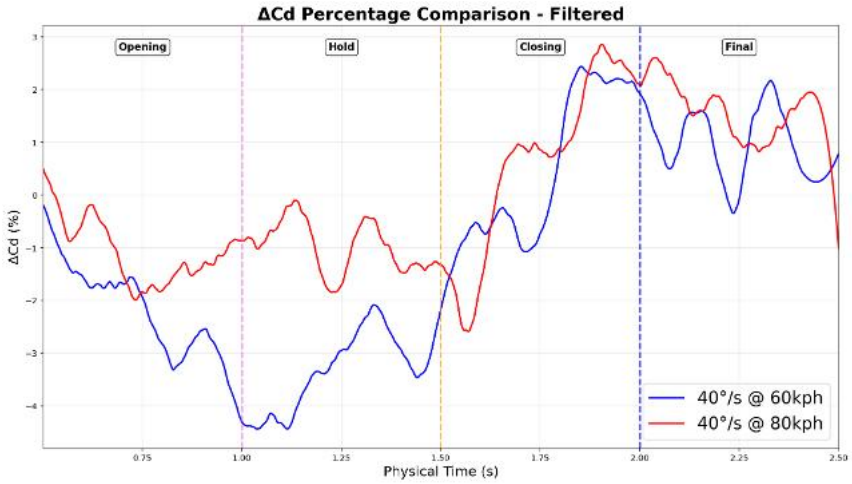
In the Figures 4.14 the normalized force coefficients of C_L and C_D can be seen. Note that C_L has a positive direction downwards, the normalization is done using the last value of the Flush phase as that is the measure for the change in the forces during the phases which the geometries are rotating.
To look into the drag and downforce difference more accurately, the percentage change of C_L and C_D will be examined.

4.3.1 Force Coefficient Percentage Change

By using the percentage change, the change in Drag and Lift during the transient phases can be seen more accurately. The RANS and Flush stages have also been removed as they do not have any useful information because it is used for the initialization. The data are also filtered using Savitzky–Golay filter, which smooths the signal by fitting successive subsets of adjacent data points with a low-degree-polynomial using least squares.



(a) ΔC_l Percentage Comparison - Filtered



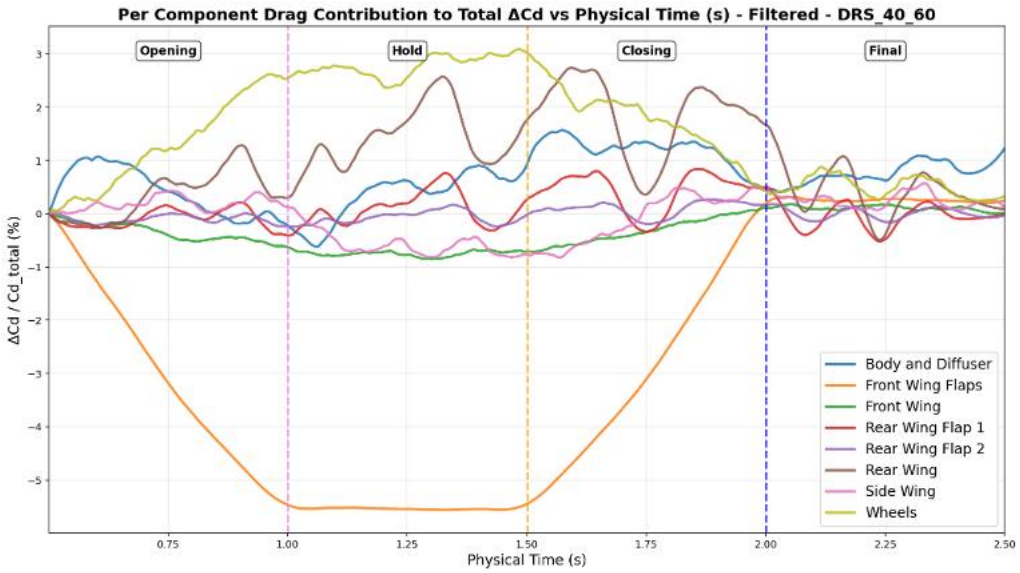
(b) ΔC_d Percentage Comparison - Filtered

Figure 4.15: Delta Force Coefficient Comparison - Filtered.

It can be extracted from the plots that for both simulation there is around a 12.5% reduction of Lift. For the drag reduction, at 60 kph there is a 4.5% reduction while for 80 kph there is a 2.6A complete Drag reduction performance will be given at Section 4.5 for all simulations.

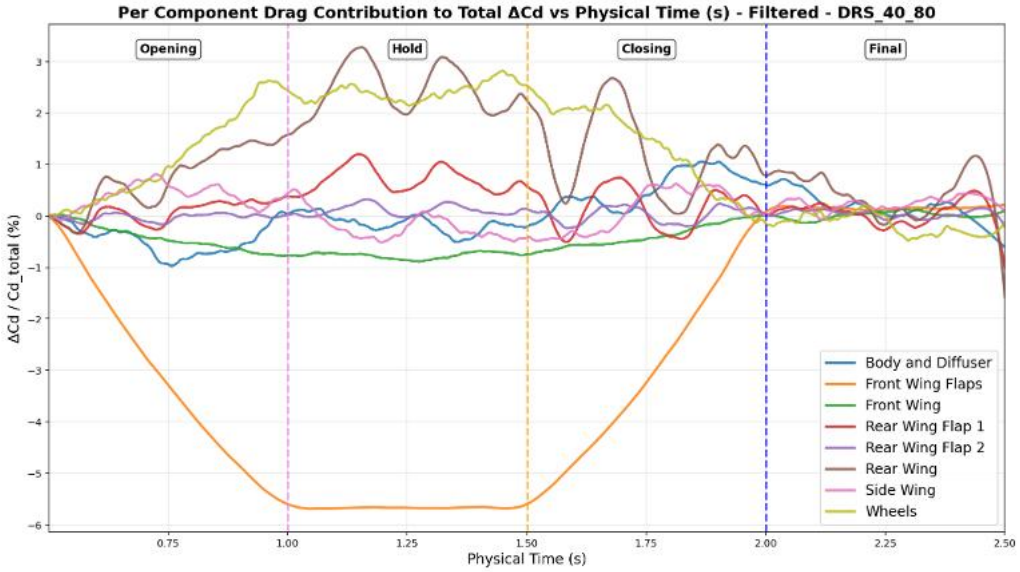
4.3.2 Per Component Drag Contribution

By focusing on the different components of the car, it can be checked which parts contribute to the overall drag reduction.



(a) DRS_40_60 Per Component ΔC_d Percentage Change

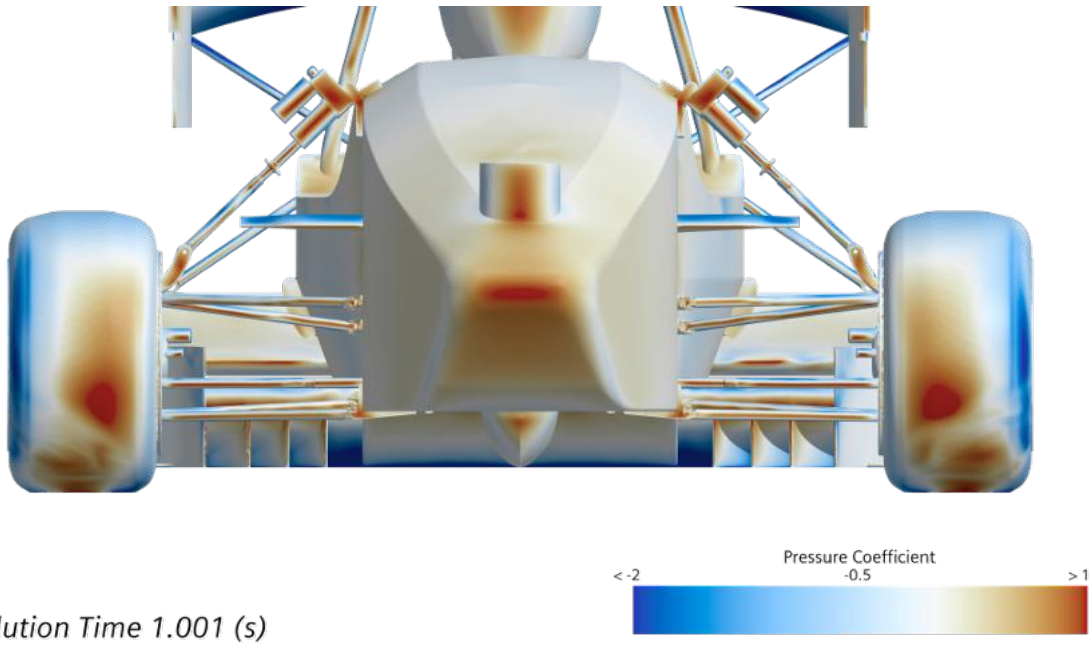
Figure 4.16: Per Component ΔC_d Percentage Change.



(b) DRS_40_80 Per Component ΔC_d Percentage Change

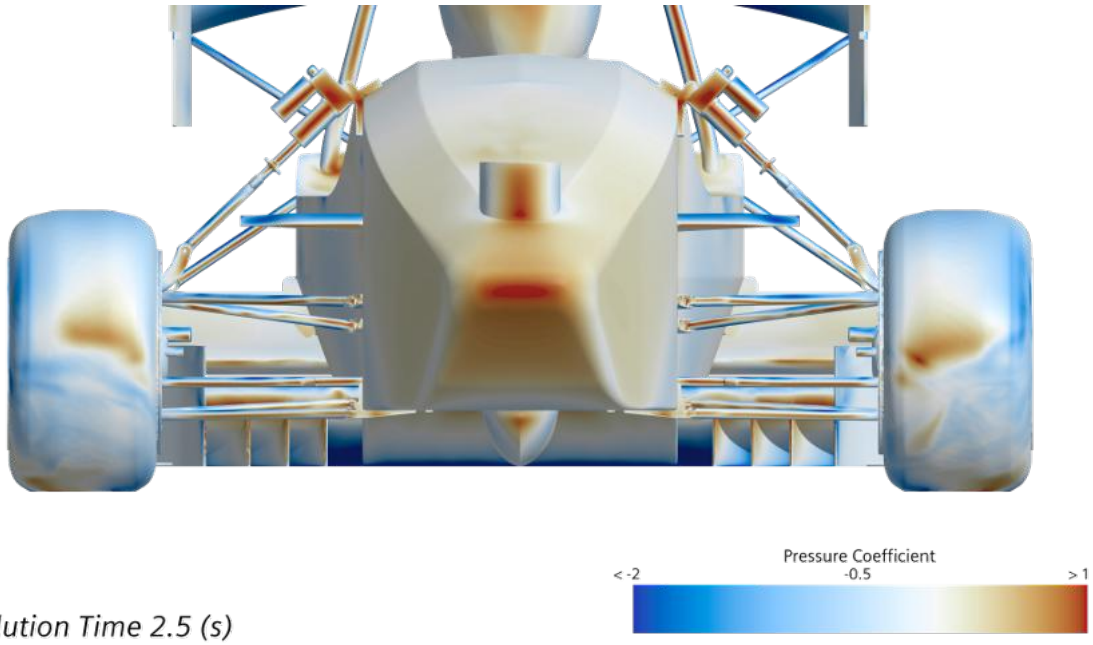
Figure 4.16: Per Component ΔC_d Percentage Change (continued).

From the per component Figures 4.16 similar reactions to the DRS Activation can be seen. The front wing flaps have substantial drag reduction and also being consistent for both speeds. Then focusing on downstream components, the components gain drag with the two main contributors being the rear wing and the wheels. With Cp scalar scenes the difference between the two Phases on the wheel leading face pressure distribution can easily be seen in Figures 4.17 where the front wing has been removed from the scene to focus on the wheel leading face pressure distribution.



(a) Surface C_p Front View Without Front Wing DRS Active

Figure 4.17: Surface C_p Front View Without Front Wing.



(b) Surface C_p Front View Without Front Wing DRS Inactive

Figure 4.17: Surface C_p Front View Without Front Wing (continued).

4.3.3 Aerodynamic Forces Conclusion

In summary, both simulations that have been used to compare forces during different speeds seem similar with the DRS_40_60 having superior performance of total drag reduction as well as a more consistent one as can be seen in Figure 4.14.There is also an important interaction between the front wing DRS system and the wheels as can be seen in Figure 4.16 and 4.3.3.

4.4 Transient Force Response: Effect of Angular Velocity

This Section examines the transient aerodynamic response during DRS actuation. The response will be analyzed for three angular velocities ($\omega = 20, 40, 80$ deg/s) at a constant vehicle speed of 80 km/h. The analysis focuses on two key metrics used in every transient analysis, overshoot and settling time. This metrics can directly impact the vehicles stability and drivability while during racing conditions. During transient effects is important to optimize the actuation rate to ensure predictable vehicle behavior. The responses of C_D, C_L and aerodynamic balance will be investigated.

4.4.1 Opening Phase Analysis

First, the focus will be on the transient responses of Phases 2 and 3 so during the opening phase until it stabilizes in the holding phase.

4.4.1.1 Overshoot During Opening

Table 4.2: Peak Overshoot During DRS Opening Phase (V = 80 km/h).

ω (deg/s)	C_L Overshoot (%)	C_D Overshoot (%)	Balance Overshoot (%)
20	1.21	0.54	0.49
40	0.31	0.97	1.18
80	0.62	0.67	1.14

As can be concluded from the table, there is no dramatic overshoot in any case with most of the cases having the overshoot within the settling time limit of 1%.

Figures C.4–C.6 illustrate the transient behavior of each aerodynamic parameter during the opening phase of DRS. The highest average overshoot is during $\omega = 40$ deg/s while the most stable is $\omega = 20$ deg/s but is very close for every angular velocity tested which makes the angular velocity overshoot during opening not a main concern for design parametrization.

4.4.1.2 Settling Time During Opening

The settling time represents the duration required for the aerodynamic forces to stabilize within a specified tolerance band (ś1% for C_L and C_D, ś0.5% for balance) after the completion of flap motion. Table 4.3 presents the measured settling times for each parameter and actuation rate that can be also seen in the Figures C.4–C.6.

Table 4.3: Settling Time After DRS Opening (V = 80 km/h).

ω (deg/s)	C_L Settling (s)	C_D Settling (s)	Balance Settling (s)
20	0.20	0.15	0.068
40	0.38	N/A	0.070
80	0.13	0.11	0.064

As shown in Figures C.4–C.6, the settling behavior varies with the specific angular velocity. It can be observed that $\omega = 80$ has the slight edge between the different rates.

4.4.2 Closing Phase Analysis

Moving on to the DRS deactivation response, focusing on Phases 4 and 5 where the forces stabilize.

4.4.2.1 Overshoot During Closing

The closing phase exhibits different transient characteristics compared to opening, as the flow reattachment process differs from separation. Table 4.4 quantifies the overshoot magnitudes during DRS closure.

Table 4.4: Peak Overshoot During DRS Closing Phase (V = 80 km/h).

ω (deg/s)	C_L Overshoot (%)	C_D Overshoot (%)	Balance Overshoot (%)
20	0.12	0.79	0.64
40	0.33	1.14	0.77
80	1.58	1.19	1.73

Figures C.1–C.3 reveal that the closing phase generally exhibits comparable overshoot magnitudes with the opening Phase. As expected the most severe transient overshoot occurs at $\omega = 80$ deg/s followed by $\omega = 40$ deg/s, suggesting that reattachment while DRS deactivates is more challenging to achieve rather when the flaps are opening.

4.4.2.2 Settling Time During Closing

The settling characteristics during DRS closing phase are quantified in Table 4.5, using the same tolerance criteria as the opening phase.

Table 4.5: Settling Time After DRS Closing (V = 80 km/h).

ω (deg/s)	C_L Settling (s)	C_D Settling (s)	Balance Settling (s)
20	0	0	0
40	0.28	0.57	0
80	0	0.12	0

As illustrated in Figures C.1–C.3, the closing phase exhibits slightly faster behavior compared to opening. Many parameters stabilize immediately upon completion of flap motion. This contradicts with the opening phase where all configurations required settling duration.

4.4.3 Conclusions

Analysis of the transient aerodynamic response across three actuation rates reveals important performance characteristics. During DRS opening, $\omega = 20$ deg/s overshoot is minimized but it costs in settling time, while $\omega = 40$ and 80 deg/s produce comparable overshoot values with faster stabilization. The closing phase demonstrates markedly superior behavior, with most parameters being settled at the end of the actuation regardless of actuation rate. The results indicate that DRS activation and deactivation poses minimal stability concerns. The optimal actuation strategy depends on whether priority is given to minimizing peak transients or reducing settling duration as well as time spent in low drag state, with this data, the focus can also be placed on variable actuation rates based on the characteristics of the track or state of the car for example braking vs accelerating.

4.5 Drag Reduction Performance: Complete Simulation Matrix

This section analyzes the drag reduction effectiveness of the DRS system across all simulated operating conditions.

4.5.1 Drag Reduction Calculation

The drag reduction percentage is calculated as:

$$\text{Drag Reduction} = \frac{C_{D,closed} - C_{D,open}}{C_{D,closed}} \times 100\% \tag{4.2}$$

Similarly, the downforce loss is:

$$\text{Downforce Loss} = \frac{|C_{L,closed}| - |C_{L,open}|}{|C_{L,closed}|} \times 100\% \tag{4.3}$$

Where the Open stage parameters are averaged throughout the holding phase and Closed parameters are averaged from final phase.

4.5.2 Drag and Downforce Reduction

Figure 4.18 presents the drag and downforce reduction achieved for each simulation case.

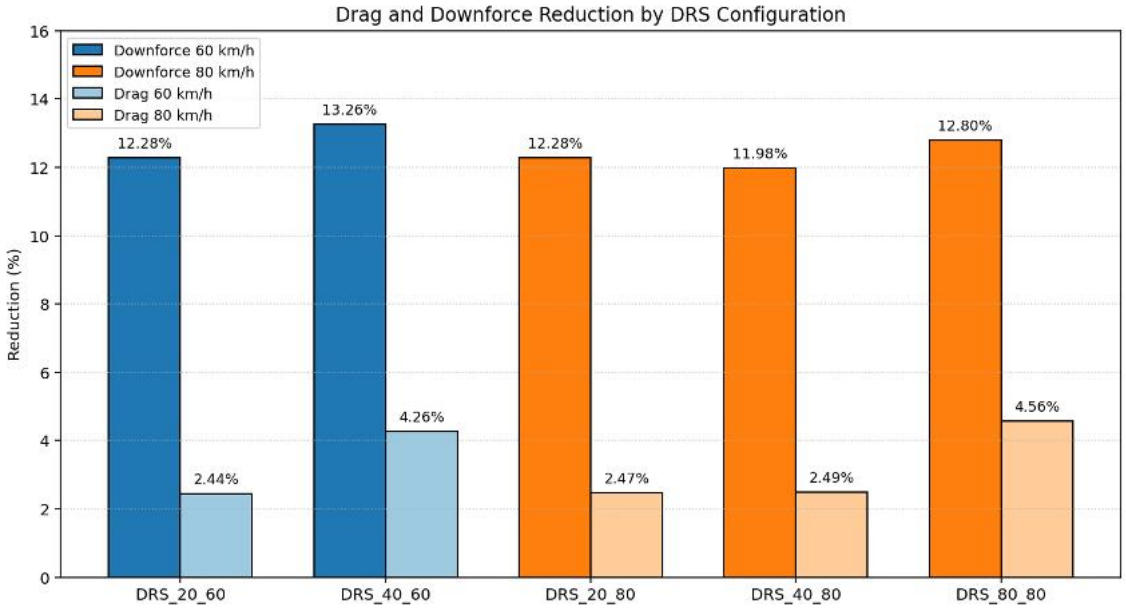


Figure 4.18: Drag Reduction Percentage for all Simulated DRS Configurations.

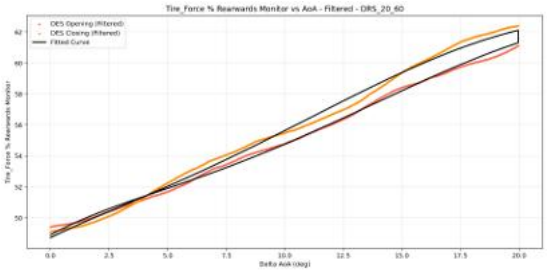
From Figures 4.18 the conclusion can be drawn that the best activation scenarios are DRS_40_60 and DRS_80_80 reaching over 4% reduction in both simulations. It will be interesting to get the results of DRS_80_60 but due to time and computational constraints it is not done. The rest of the simulations perform about the same every one reaching around 2.5% of drag reduction

4.6 Effects on Vehicle Dynamics

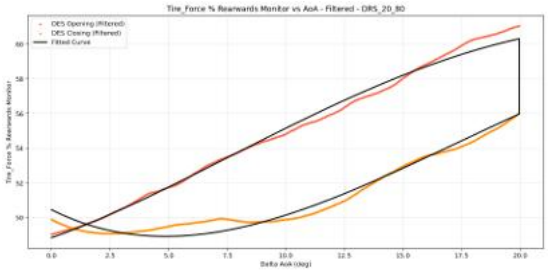
In this section, the focus will be on how the angle of attack of the flaps in the closing and opening phase affect the balance of the car. Hysteresis effects are expected, which will be similar to the effects seen during pitch analysis.

4.6.1 Hysteresis Based on Speed and Actuation Rate

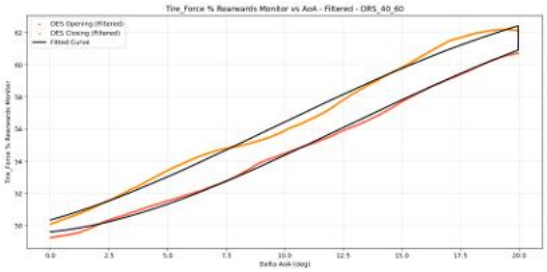
The data are filtered and a 4th degree polynomial curve is fitted to them. The equations of the fitted curves are given in Appendix B.



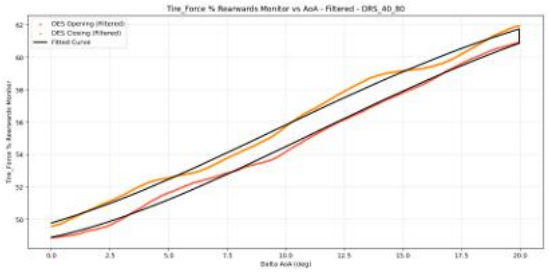
(a) DRS_20_60



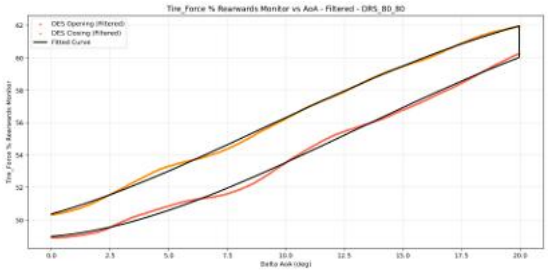
(b) DRS_20_80



(c) DRS_40_60



(d) DRS_40_80



(e) DRS_80_80

Figure 4.19: Vehicle Balance (Tire Force % Rearwards) vs Angle of Attack (AoA) for Various DRS Configurations.

From the plots, the balance of the car can be seen while the flaps are opening (Orange Lines) and while closing (Yellow Line) and the fitted polynomial curve (Black Lines). As expected a hysteresis is formed, that can help to further predict the stability of the car and if changes are needed, the fitted curves polynomial can be found in Appendix D.
Major differences of the different actuation is that at $\omega = 20$ deg/s the balance reaches 60% while for the cases of $\omega = 40$ deg/s and $\omega = 80$ deg/s the balance reaches up to 62% rear wards. Another difference between $\omega = 40$ deg/s and $\omega = 80$ deg/s is the range difference between the closing and opening phase with the 80 degrees being wider so the response is more unpredictable.
Overall the balance at 0 degrees has an average of 1% difference. Considering that the closing of the flaps might be done as close to the braking zone as possible as long as the DRS is fully inactive during braking.

5
Conclusion

This chapter presents a summary of the results followed by a discussion over the application of the system as well as limitations that the thesis had and potential future work.

5.1 Aerodynamic Performance

The aerodynamic performance of the DRS system was quantified to enable educated and data driven decisions if an active DRS system will add substantial gains to overcome the added complication and weight gain to the front wing assembly. Regarding aerodynamic performance, the flow field and the force generation on different parts of the car were investigated. The presented data show that a general drag reduction of 4.5% can be achieved with a non-optimized DRS mechanism. Applying this drag reduction to the acceleration event yields a reduction in run time of 0.006 s. Further gains could be expected in the endurance event, where the reduced power requirement would improve the score in the efficiency event. However, this requires further investigation using a speed-dependent drag coefficient, and the additional mass of the activation mechanism must also be accounted for. Overall, the performance of the DRS was efficient for a small part of the car, also some more optimization can be implemented for example the outer flaps not opening as shown in Figures 4.17 all of the drag being reduced by that part of the flaps is being regained by the wheels. Another part that was generating more drag while the DRS was activated was the rear wing, this leads to the conclusion that a DRS system that rotates both front and rear wing geometries will be more efficient.

5.2 Aerodynamic Transient Response

For the transient analysis, a good amount of angles where tested and differences were found between them, with the faster actuation being superior both because it reduces the drag component faster than the others so more time is spent in the low drag setup as well as the transient responses that were evaluated. It will be interesting to look into even quicker actuation but due to limitations this can be done in future work. Focusing more on the effects of this transient phenomenon to the vehicle dynamics, stable behaviors are observed both in transient parameters like overshoot and settling time as well as load transfer from front to rear axle. This leads to the conclusion that the car is stable car even when DRS is being activated or deactivated in acceleration and braking zones as well as turns with big radius

that the vehicle speed is higher than 60 kph this can also be implemented on the control system side where more parameters than the speed can be evaluated for the activation or deactivation of DRS like steering wheel angle or slip angle calculated by other sensors of the car.

5.3 Final Verdict

Overall the system can be applied but further investigation should be done focusing on mass gain drawback as well as complexity added to the assembly against the point gain potential in dynamic and static events. Further investigation can be done in steady state conditions to reduce computational power needed and to find an optimal drag reduction state as the transient analysis should not be considered as a major design parameter as concluded in the Results Chapter 4 as long as the velocities used have been tested. Another conclusion that was reached is that an active rear wing will also be beneficial, this can increase the efficiency of the DRS that has been investigated for the front wing. This has been a good start for the introduction of transient aerodynamics for the team, developing and testing methodologies that can be used for the future development of the car.

5.4 Limitations and Future Work

The biggest limitation of the thesis was the computational time and power needed, with simulations taking up to three days from meshing to simulation without taking postprocessing into account. Another limitation was the mesh size, during this simulations the limit of allowed cells was reached due to the available memory of the computer, cluster processing could have been used during the meshing but having a bigger mesh will lead to instability while postprocessing on the available computer. Another limitation is the validation of the data, testing a new wing that has the capability to use the DRS mechanism will be expensive and time consuming, which during the short period of the thesis is not feasible. For future work, there are many interesting sectors that can be explored. Starting with the geometry a more refined mechanism can be designed, with the actuation, hydraulics, routing and weights being assigned to each part for more accurate lap time simulations and evaluation of the gains. Continuing to the simulation, the rear wing can also be simulated with active DRS as the simulation file is already set up for it, another interesting simulation scenario other than faster actuation is asymmetric actuation during cornering effecting the lateral balance of the car. Another part of the DRS system that has not been worked on is the control system which will act as the brain of the DRS and operate when needed so that the car is working at the optimal Phase during racing conditions. Finally for validation, a prototype wing can be made with cheaper material then the actual parts like glass fiber and hopefully tested in the wind tunnel.

Bibliography

[1] John D. Anderson. *Fundamentals of Aerodynamics*. 6th. Essential textbook covering boundary layers, inviscid flow, viscous flow, and fundamental principles. McGraw-Hill Education, 2017.

[2] Peiqing Liu. *Aerodynamics*. 2nd. Springer, 2022. ISBN: 978-981-19-4585-1.

[3] Hermann Schlichting and Klaus Gersten. *Boundary-Layer Theory*. 9th. The definitive reference on boundary layer physics and separation. Springer, 2017.

[4] Philip G. Saffman. *Vortex Dynamics*. Fundamental theory of vortex formation and evolution. Cambridge University Press, 1992.

[5] Ismet Gursul. "Review of unsteady vortex flows over slender delta wings". In: *Journal of Aircraft* 42.2 (2005). Vortex breakdown and unsteady vortical flows, pp. 299–319.

[6] Joseph Katz and Allen Plotkin. *Low-Speed Aerodynamics*. 2nd. Comprehensive treatment of vortex methods, panel methods, and low-speed wing theory. Cambridge University Press, 2001.

[7] Joseph Katz. *Race Car Aerodynamics: Designing for Speed*. Classic reference on automotive and race car aerodynamics. Bentley Publishers, 1995.

[8] Xin Zhang and Jonathan Zerihan. "Aerodynamics of Gurney flaps on a wing in ground effect". In: *AIAA Journal* 41.6 (2003). Effect of Gurney flaps on downforce-generating wings, pp. 1109–1112.

[9] Jonathan Zerihan and Xin Zhang. "Aerodynamics of a single element wing in ground effect". In: *Journal of Aircraft* 37.6 (2000). Fundamental study of wings in ground effect - crucial for Formula Student, pp. 1058–1064.

[10] Simon McBeath. *Competition Car Aerodynamics: A Practical Handbook*. 3rd. Practical guide to race car aerodynamic design and development. Veloce Publishing, 2017.

[11] Siemens Digital Industries Software. *External Aerodynamics with Simcenter STAR-CCM+: Best Practice Guidelines*. Version 2502. Siemens. 2025.

[12] Siemens Digital Industries Software. *How to choose a time-step for a motion simulation using overset mesh*. Knowledge Base Article KB000031791. Accessed: 2026-06-12. URL: `https://support.sw.siemens.com/en-US/knowledge-base/KB000031791_EN_US`.

A

Appendix: Simulation Model Coefficients

Table A.1: SST k-ω and IDDES Model Constants.

Parameter	**Symbol**	**Value**
SST Model Constants (RANS & DES)		
Von Kármán constant	κ	0.41
Turbulent kinetic energy destruction	β^*	0.09
Blending coefficient	a_1	0.31
Inner region destruction	β_1	0.075
Inner region k diffusion	σ_{k1}	0.85
Inner region ω diffusion	$\sigma_{\omega 1}$	0.5
Outer region destruction	β_2	0.0828
Outer region k diffusion	σ_{k2}	1.0
Outer region ω diffusion	$\sigma_{\omega 2}$	0.856
IDDES-Specific Constants (DES only)		
DES constant (k-ω region)	$C_{des,Kw}$	0.78
DES constant (k-ϵ region)	$C_{des,Ke}$	0.61
DES time limiter	$C_{des,TimeLim}$	1.0

B

Appendix: Postprocessing Python Scripts

B.1 Data Analysis Script

```python
"""
Created on Tue Apr 14 10:57:17 2026

@author: Panos Papakostas
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
import os
from scipy.signal import savgol_filter
from settling_analysis import settling_time_analysis

#Chose Simulation to Process
os.chdir(r'C:/Users/panay/Desktop/Thesis/40degs@60/CSV')

#Verify directory
print("Current directory:", os.getcwd())

# Import the CSV file
df = pd.read_csv("Exported_data.csv")

# Remove all rows that contain any NaN values
df_cleaned = df.dropna().copy()

#==============================================================================
# %% SIMULATION CONFIGURATION
#==============================================================================
sim_config = {
    "RANS":                 {"start": 0,     "end": 5000},
    "Flush":                {"start": 5001,  "end": 9000},
    "DES_flaps_opening":    {"start": 9001,  "end": 13000},
    "DES_no_movement":      {"start": 13001, "end": 17000},
```

```
    "DES_flaps_closing":       {"start": 17001,  "end": 21000},
    "DES_Final":               {"start": 21001,  "end": 25000},
}

#
↪ ============================================================================================
# SIMULATION PARAMETERS
#
↪ ============================================================================================
dt = 1E-3                     # seconds per timestep
omega_deg = 40                # deg/s
iters_per_timestep = 8        # iterations per timestep
deg_per_timestep = omega_deg * dt  # = 0.04 deg per timestep

#
↪ ============================================================================================
# CALCULATE DAoA (Delta Angle of Attack)
#
↪ ============================================================================================
df_cleaned['DAoA'] = 0.0

for idx in df_cleaned.index:
    iter_num = df_cleaned.loc[idx, 'Iteration']

    if 0 <= iter_num <= 5000:  # RANS
        df_cleaned.loc[idx, 'DAoA'] = 0.0

    elif 5001 <= iter_num <= 9000:  # Flush
        df_cleaned.loc[idx, 'DAoA'] = 0.0

    elif 9001 <= iter_num <= 13000:  # DES_flaps_opening
     timestep_number = (iter_num - 9001) // iters_per_timestep  # Integer
     ↪ division
     df_cleaned.loc[idx, 'DAoA'] = timestep_number * deg_per_timestep

    elif 13001 <= iter_num <= 17000:  # DES_no_movement
        # Hold at maximum deflection
        max_deflection = (17000 - 13001) / iters_per_timestep *
        ↪ deg_per_timestep
        df_cleaned.loc[idx, 'DAoA'] = max_deflection

    elif 17001 <= iter_num <= 21000:  # DES_flaps_closing
        max_deflection = (21000 - 17001) / iters_per_timestep *
        ↪ deg_per_timestep
        step_number = (iter_num - 17001) / iters_per_timestep
        df_cleaned.loc[idx, 'DAoA'] = max_deflection - (step_number *
        ↪ deg_per_timestep)

    elif 21001 <= iter_num <= 25000:  # DES_Final
        df_cleaned.loc[idx, 'DAoA'] = 0.0
```

```
77
78  #
    ↪  ============================================================================================
79  # %% SPLIT INTO PHASES
80  #
    ↪  ============================================================================================
81  splits = {}
82
83  # RANS: split from original df (before cleaning)
84  mask_rans = (df["Iteration"] >= sim_config["RANS"]["start"]) &
    ↪  (df["Iteration"] <= sim_config["RANS"]["end"])
85  splits["RANS"] = df[mask_rans].reset_index(drop=True)
86
87  # All other splits: from cleaned df
88  for name, bounds in sim_config.items():
89      if name == "RANS":
90          continue  # Already handled above
91      mask = (df_cleaned["Iteration"] >= bounds["start"]) &
        ↪  (df_cleaned["Iteration"] <= bounds["end"])
92      splits[name] = df_cleaned[mask].reset_index(drop=True)
93      print(f"{name}: {len(splits[name])} rows")
94
95  # Access individual dataframes
96  df_rans = splits["RANS"]
97  df_flush = splits["Flush"]
98  df_des_opening = splits["DES_flaps_opening"]
99  df_des_no_movement = splits["DES_no_movement"]
100 df_des_closing = splits["DES_flaps_closing"]
101 df_des_final = splits["DES_Final"]
102
103 # Combined DES dataframe
104 df_des = pd.concat([df_flush, df_des_opening, df_des_no_movement,
    ↪  df_des_closing, df_des_final], axis=0)
105 df_des = df_des.reset_index(drop=True)
106
107 ## Normalized dataframe
108 # Create a normalized copy of df_cleaned (DES data)
109 df_des_normalized = df_cleaned.copy()
110
111 # Create a normalized copy of df_rans (RANS data)
112 df_rans_normalized = df_rans.copy()
113
114 # Combine RANS + DES normalized data
115 df_normalized = pd.concat([df_rans_normalized, df_des_normalized], axis=0)
116 df_normalized = df_normalized.reset_index(drop=True)
117
118 # List of columns to exclude from normalization (keep as-is)
119 exclude_columns = ['Iteration', 'DAoA','Physical Time (s)']  # Add any
    ↪  other columns you want to keep unchanged
120
```

```python
# Get all numeric columns except the excluded ones
columns_to_normalize = [col for col in df_cleaned.columns
                        if col not in exclude_columns and
                        ↪  pd.api.types.is_numeric_dtype(df_cleaned[col])]

# Get the last value of the Flush phase (iteration 9000 is the last flush
↪  iteration)
flush_reference_idx = df_normalized[df_normalized['Iteration'] <=
↪  9000].index[-1]

# Normalize each column relative to the last flush value
for col in columns_to_normalize:
    reference_value = df_normalized.loc[flush_reference_idx, col]

    if reference_value != 0:
       df_normalized[col] = df_normalized[col] / reference_value
    else:
        df_normalized[col] = 0

print("Normalized relative to last Flush value (Iteration 9000)")

#
↪  ============================================================================================
# %% NOISE REMOVAL - FILTERING
#
↪  ============================================================================================

#Drag columns definition
drag_columns = [
    'Drag - Body and Diffuser (NC) Monitor (N)',
    'Drag - Front Wing Flaps (NC) Monitor (N)',
    'Drag - Front Wing (NC) Monitor (N)',
    'Drag - Rear Wing Flap 1 (NC) Monitor (N)',
    'Drag - Rear Wing Flap 2 (NC) Monitor (N)',
    'Drag - Rear Wing (NC) Monitor (N)',
    'Drag - Side Wing (NC) Monitor (N)',
    'Drag - Wheels Monitor (N)'
]

def apply_moving_average(data, window_size=50):
    """Apply moving average filter"""
    return data.rolling(window=window_size, center=True).mean()

def apply_savgol(data, window_length=51, polyorder=3):
    """Apply Savitzky-Golay filter"""
    return savgol_filter(data, window_length=window_length,
    ↪  polyorder=polyorder)

# Apply filters to cleaned dataframe
window_size = 180  # Adjust as needed
```

```
savgol_window = 181  # Must be odd number
savgol_poly = 3

# Filter drag columns
for col in drag_columns:
    df_cleaned[f'{col}_MA'] = apply_moving_average(df_cleaned[col], window_size)
    df_cleaned[f'{col}_SG'] = apply_savgol(df_cleaned[col], savgol_window, savgol_poly)

# Filter total drag and downforce
df_cleaned['Drag - Total (NC) Monitor (N)_MA'] = apply_moving_average(df_cleaned['Drag - Total (NC) Monitor (N)'], window_size)
df_cleaned['Drag - Total (NC) Monitor (N)_SG'] = apply_savgol(df_cleaned['Drag - Total (NC) Monitor (N)'], savgol_window, savgol_poly)

df_cleaned['Downforce - Total Monitor_MA'] = apply_moving_average(df_cleaned['Downforce - Total Monitor'], window_size)
df_cleaned['Downforce - Total Monitor_SG'] = apply_savgol(df_cleaned['Downforce - Total Monitor'], savgol_window, savgol_poly)

# Filter Cd and Cl
df_cleaned['Cd Monitor_MA'] = apply_moving_average(df_cleaned['Cd Monitor'], window_size)
df_cleaned['Cd Monitor_SG'] = apply_savgol(df_cleaned['Cd Monitor'], savgol_window, savgol_poly)

df_cleaned['Cl Monitor_MA'] = apply_moving_average(df_cleaned['Cl Monitor'], window_size)
df_cleaned['Cl Monitor_SG'] = apply_savgol(df_cleaned['Cl Monitor'], savgol_window, savgol_poly)

# Filter balance/tire force
df_cleaned['Tire_Force % Rearwards Monitor_MA'] = apply_moving_average(df_cleaned['Tire_Force % Rearwards Monitor'], window_size)
df_cleaned['Tire_Force % Rearwards Monitor_SG'] = apply_savgol(df_cleaned['Tire_Force % Rearwards Monitor'], savgol_window, savgol_poly)

# Fill NaN values in moving average columns with original values
for col in drag_columns:
    df_cleaned[f'{col}_MA'].fillna(df_cleaned[col], inplace=True)

df_cleaned['Drag - Total (NC) Monitor (N)_MA'].fillna(df_cleaned['Drag - Total (NC) Monitor (N)'], inplace=True)
```

```
196 df_cleaned['Downforce - Total Monitor_MA'].fillna(df_cleaned['Downforce -
    ↪ Total Monitor'], inplace=True)
197
198 df_cleaned['Cd Monitor_MA'].fillna(df_cleaned['Cd Monitor'], inplace=True)
199 df_cleaned['Cl Monitor_MA'].fillna(df_cleaned['Cl Monitor'], inplace=True)
200
201 df_cleaned['Tire_Force % Rearwards
    ↪ Monitor_MA'].fillna(df_cleaned['Tire_Force % Rearwards Monitor'],
    ↪ inplace=True)
202
203 #print("Filtering complete - _MA and _SG columns added to df_cleaned")
204
205 df_cleaned['Drag - Front Wing (NC) Monitor (N)_SG_corrected'] = (
206     df_cleaned['Drag - Front Wing (NC) Monitor (N)_SG'] -
207     df_cleaned['Drag - Front Wing Flaps (NC) Monitor (N)_SG']
208 )
209
210 df_des_opening = df_cleaned[
211     (df_cleaned["Iteration"] >= 9001) & (df_cleaned["Iteration"] <= 13000)
212 ].reset_index(drop=True)
213
214 df_des_closing = df_cleaned[
215     (df_cleaned["Iteration"] >= 17001) & (df_cleaned["Iteration"] <=
        ↪ 21000)
216 ].reset_index(drop=True)
217
218 # %% Curve fit setup
219 #=============================================================================================
220 # Curve fitting balance - USE FILTERED DATA
221 t = np.concatenate([
222     df_des_opening["Physical Time (s)"].values,
223     df_des_closing["Physical Time (s)"].values])
224
225 # USE FILTERED BALANCE DATA
226 y_cop = np.concatenate([
227     df_des_opening["Tire_Force % Rearwards Monitor_SG"].values,
228     df_des_closing["Tire_Force % Rearwards Monitor_SG"].values])
229
230 aoa = np.concatenate([
231     df_des_opening["DAoA"].values,
232     df_des_closing["DAoA"].values
233 ])
234
235 idx = np.argsort(t)
236 t = t[idx]
237 y_cop = y_cop[idx]
238 aoa = aoa[idx]
239
240 # REDUCE polynomial order for better fit
241 p_t_cop = np.polyfit(t, y_cop, 4)
```

```
242 y_fit_t_cop = np.polyval(p_t_cop, t)
243
244 ##IF YOU WANT TO PRINT POLYNOMIAL TO LATEX
245 x = sp.symbols('x')
246 poly_expr = sum(sp.Float(coef)*x**i for i, coef in
    ↪ enumerate(p_t_cop[::-1]))
247 print(poly_expr)          # symbolic form
248 print(sp.latex(poly_expr))  # LaTeX form
249
250 # Curve fit Cl - USE FILTERED DATA
251 # Time and AoA taken from previous curve fitting
252 y_Cl = np.concatenate([
253     df_des_opening["Cl Monitor_SG"].values,
254     df_des_closing["Cl Monitor_SG"].values])
255
256 idx = np.argsort(t)
257 t = t[idx]
258 y_Cl = y_Cl[idx]
259 aoa = aoa[idx]
260
261 p_t_cl = np.polyfit(t, y_Cl, 4)
262 y_fit_t_Cl = np.polyval(p_t_cl, t)
263
264 # Curve fit Cd - USE FILTERED DATA
265 # Time and AoA taken from previous curve fitting
266 y_Cd = np.concatenate([
267     df_des_opening["Cd Monitor_SG"].values,
268     df_des_closing["Cd Monitor_SG"].values])
269
270 idx = np.argsort(t)
271 t = t[idx]
272 y_Cd = y_Cd[idx]
273 aoa = aoa[idx]
274
275 p_t_cd = np.polyfit(t, y_Cd, 6)
276 y_fit_t_Cd = np.polyval(p_t_cd, t)
277
278
279 # %% PLOTS
280 #=============================================================================================
281 ## Physical time splits
282 # Get the physical times at each transition point (use the first available
    ↪ iteration in each range)
283 time_flush_start = df_cleaned[df_cleaned['Iteration'] >= 5001]['Physical
    ↪ Time (s)'].iloc[0]
284 time_opening_start = df_cleaned[df_cleaned['Iteration'] >= 9001]['Physical
    ↪ Time (s)'].iloc[0]
285 time_no_movement_start = df_cleaned[df_cleaned['Iteration'] >=
    ↪ 13001]['Physical Time (s)'].iloc[0]
```

```
time_closing_start = df_cleaned[df_cleaned['Iteration'] >= 17001]['Physical Time (s)'].iloc[0]
time_final_start = df_cleaned[df_cleaned['Iteration'] >= 21001]['Physical Time (s)'].iloc[0]

#Cd
fig, ax = plt.subplots(figsize=(12, 6))

#ax.scatter(df_rans["DAoA"],          df_rans["Cd Monitor"], color="steelblue", s=5, label="RANS")
ax.scatter(df_des_opening["DAoA"],  df_des_opening["Cd Monitor"], color="tomato",    s=5, label="DES Opening")
ax.scatter(df_des_no_movement["DAoA"], df_des_no_movement["Cd Monitor"], color="seagreen",  s=5, label="DES No Movement")
ax.scatter(df_des_closing["DAoA"],  df_des_closing["Cd Monitor"], color="darkorange", s=5, label="DES Closing")
ax.plot(aoa, y_fit_t_Cd, color="black", linewidth=2, label="Fitted Curve")

ax.set_xlabel("Delta AoA (deg)")
ax.set_ylabel("Cd Monitor")
ax.set_title("Cd vs AoA - All Stages")
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.margins(x=0+0.01)
plt.show()

#Balance
fig, ax = plt.subplots(figsize=(12, 6))

# Plot filtered data
ax.scatter(df_des_opening["DAoA"],  df_des_opening["Tire_Force % Rearwards Monitor_SG"],  color="tomato",    s=5, label="DES Opening (Filtered)")
ax.scatter(df_des_closing["DAoA"],  df_des_closing["Tire_Force % Rearwards Monitor_SG"],  color="darkorange", s=5, label="DES Closing (Filtered)")
ax.plot(aoa, y_fit_t_cop, color="black", linewidth=2, label="Fitted Curve")

ax.set_xlabel("Delta AoA (deg)")
ax.set_ylabel("Tire_Force % Rearwards Monitor")
ax.set_title("Tire_Force % Rearwards Monitor vs AoA - Filtered - DRS_40_60")
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.margins(x=0)
ax.set_xlim(-1, 21)
plt.show()

```

```
325 #Cl
326 fig, ax = plt.subplots(figsize=(12, 6))
327
328 #ax.scatter(df_rans["DAoA"],          df_rans["Cd Monitor"],
    ↪  color="steelblue", s=5, label="RANS")
329 ax.scatter(df_des_opening["DAoA"],  df_des_opening["Cl Monitor"],
    ↪  color="tomato",    s=5, label="DES Opening")
330 #ax.scatter(df_des_no_movement["DAoA"], df_des_no_movement["Cl Monitor"],
    ↪  color="seagreen",  s=5, label="DES No Movement")
331 ax.scatter(df_des_closing["DAoA"],  df_des_closing["Cl Monitor"],
    ↪  color="darkorange", s=5, label="DES Closing")
332 ax.plot(aoa, y_fit_t_Cl, color="black", linewidth=2, label="Fitted Curve")
333
334 ax.set_xlabel("Delta AoA (deg)")
335 ax.set_ylabel("Cl")
336 ax.set_title("Cl vs AoA")
337 ax.legend()
338 ax.grid(True, alpha=0.3)
339 plt.tight_layout()
340 plt.margins(x=0)
341 plt.show()
342
343 #Drag Analysis
344 fig, ax = plt.subplots(figsize=(14, 8))
345
346 colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
    ↪  '#8c564b', '#e377c2', '#bcbd22']
347
348 for i, col in enumerate(drag_columns):
349     label = col.replace(' Monitor (N)', '').replace('Drag - ', '')
350     ax.plot(df_cleaned["Physical Time (s)"], df_cleaned[col],
351             color=colors[i],
352             linewidth=1.5,
353             label=label,
354             alpha=0.85)
355 # Add vertical lines
356 plt.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
357 plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
358 plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
359 plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
360
361 # Add annotations
362 y_position = plt.ylim()[1] * 0.9  # Position at 90% of y-axis height
363
364 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
365          ha='center', va='top', fontsize=12, fontweight='bold',
```

```
366             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
367
368 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪  'Opening',
369             ha='center', va='top', fontsize=12, fontweight='bold',
370             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
371
372 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪  'Hold',
373             ha='center', va='top', fontsize=12, fontweight='bold',
374             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
375
376 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
377             ha='center', va='top', fontsize=12, fontweight='bold',
378             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
379
380 time_end = df_cleaned['Physical Time (s)'].iloc[-1]
381 plt.text((time_final_start + time_end)/2, y_position, 'Final',
382             ha='center', va='top', fontsize=12, fontweight='bold',
383             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
384
385 ax.set_xlabel("Physical Time (s)", fontsize=12)
386 ax.set_ylabel("Drag (N)", fontsize=12)
387 ax.set_title("Per Component Drag vs Physical Time (s)", fontsize=14,
    ↪  fontweight='bold')
388 # Place legend to the right of the plot
389 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
390 ax.grid(True, alpha=0.3)
391 plt.tight_layout()
392 plt.margins(x=0)
393 plt.show()
394
395 #Normalized
396 df_percent = (df_cleaned / df_des_opening.iloc[0] - 1) * 100
397
398 # Create filtered percentage dataframe for Cd and Cl
399 df_percent['Cd Monitor_SG'] = ((df_cleaned['Cd Monitor_SG'] /
    ↪  df_des_opening.iloc[0]['Cd Monitor'] - 1) * 100)
400 df_percent['Cl Monitor_SG'] = ((df_cleaned['Cl Monitor_SG'] /
    ↪  df_des_opening.iloc[0]['Cl Monitor'] - 1) * 100)
401
402 fig, ax = plt.subplots(figsize=(14, 8))
403
404 colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
    ↪  '#8c564b', '#e377c2', '#bcbd22']
405
406 for i, col in enumerate(drag_columns):
407     label = col.replace(' Monitor (N)', '').replace('Drag - ', '')
408     ax.plot(df_cleaned["Physical Time (s)"], df_percent[col],
409             color=colors[i],
```

```
410             linewidth=1.5,
411             label=label,
412             alpha=0.85)
413 # Add vertical lines
414 plt.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
415 plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
416 plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
417 plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
418
419 # Add annotations
420 y_position = plt.ylim()[1] * 0.9  # Position at 90% of y-axis height
421
422 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
423          ha='center', va='top', fontsize=12, fontweight='bold',
424          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
425
426 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪  'Opening',
427          ha='center', va='top', fontsize=12, fontweight='bold',
428          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
429
430 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪  'Hold',
431          ha='center', va='top', fontsize=12, fontweight='bold',
432          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
433
434 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
435          ha='center', va='top', fontsize=12, fontweight='bold',
436          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
437
438 time_end = df_cleaned['Physical Time (s)'].iloc[-1]
439 plt.text((time_final_start + time_end)/2, y_position, 'Final',
440          ha='center', va='top', fontsize=12, fontweight='bold',
441          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
442
443 ax.set_xlabel("Physical Time (s)", fontsize=12)
444 ax.set_ylabel("Drag (%)", fontsize=12)
445 ax.set_title("Normalized Per Component D vs Physical Time (s)",
    ↪  fontsize=14, fontweight='bold')
446 ax.legend(loc='best', framealpha=0.9, fontsize=10)
447 ax.grid(True, alpha=0.3)
448 plt.tight_layout()
449 plt.margins(x=0)
450 plt.show()
451
452 #Normalized drag
```

```
453 fig, ax = plt.subplots(figsize=(14, 8))
454
455 ax.plot(df_cleaned["Physical Time (s)"], df_percent["Cd Monitor"],
    ↪  color="black", linewidth=2, label="Normalized Drag")
456 # Add vertical lines
457 plt.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
458 plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
459 plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
460 plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
461
462 # Add annotations
463 y_position = plt.ylim()[1] * 0.9  # Position at 90% of y-axis height
464
465 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
466          ha='center', va='top', fontsize=12, fontweight='bold',
467          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
468
469 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪  'Opening',
470          ha='center', va='top', fontsize=12, fontweight='bold',
471          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
472
473 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪  'Hold',
474          ha='center', va='top', fontsize=12, fontweight='bold',
475          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
476
477 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
478          ha='center', va='top', fontsize=12, fontweight='bold',
479          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
480
481 time_end = df_cleaned['Physical Time (s)'].iloc[-1]
482 plt.text((time_final_start + time_end)/2, y_position, 'Final',
483          ha='center', va='top', fontsize=12, fontweight='bold',
484          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
485
486
487 ax.set_xlabel("Physical Time (s)", fontsize=14)
488 ax.set_ylabel("Cd (%)", fontsize=14)
489 ax.set_title("Cd percentage", fontsize=16, fontweight='bold')
490 ax.legend(loc='best', framealpha=0.9, fontsize=10)
491 ax.grid(True, alpha=0.3)
492 plt.tight_layout()
493 plt.margins(x=0)
494 plt.show()
495
```

```
#Normalized Downforce
fig, ax = plt.subplots(figsize=(14, 8))

ax.plot(df_cleaned["Physical Time (s)"], df_percent["Cl Monitor"],
    color="black", linewidth=2, label="Normalized Downforce")

# Add vertical lines
plt.axvline(x=time_opening_start, color='green', linestyle='--',
    linewidth=2, alpha=0.7)
plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    linewidth=2, alpha=0.7)
plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    linewidth=2, alpha=0.7)
plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    alpha=0.7)

# Add annotations
y_position = plt.ylim()[1] * 0.9  # Position at 90% of y-axis height

plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
         ha='center', va='top', fontsize=12, fontweight='bold',
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    'Opening',
         ha='center', va='top', fontsize=12, fontweight='bold',
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    'Hold',
         ha='center', va='top', fontsize=12, fontweight='bold',
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
         ha='center', va='top', fontsize=12, fontweight='bold',
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

time_end = df_cleaned['Physical Time (s)'].iloc[-1]
plt.text((time_final_start + time_end)/2, y_position, 'Final',
         ha='center', va='top', fontsize=12, fontweight='bold',
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))


ax.set_xlabel("Physical Time (s)", fontsize=14)
ax.set_ylabel("Cl (%)", fontsize=14)
ax.set_title("Cl percentage", fontsize=16, fontweight='bold')
# ax.legend(loc='best', framealpha=0.9, fontsize=10)
ax.grid(True, alpha=0.3)
# plt.tight_layout()
plt.margins(x=0)
```

```
539 plt.show()
540
541 #RANS AND DES
542 fig, ax = plt.subplots(figsize=(14, 8))
543
544 ax.plot(df_normalized["Iteration"], df_normalized["Cl Monitor"],
    ↪ color="black", linewidth=2, label="Normalized Downforce")
545
546 plt.axvline(x=5000, color='red', linestyle='--', linewidth=3, alpha=0.7)
547 plt.axvline(x=9000, color='green', linestyle='--', linewidth=3, alpha=0.7)
548 plt.axvline(x=13000, color='purple', linestyle='--', linewidth=3,
    ↪ alpha=0.7)
549 plt.axvline(x=17000, color='orange', linestyle='--', linewidth=3,
    ↪ alpha=0.7)
550 plt.axvline(x=21000, color='blue', linestyle='--', linewidth=3, alpha=0.7)
551
552 # Add annotations
553 y_position = plt.ylim()[1] * 0.1  # Position at 10% of y-axis height
554
555
556 plt.text(2500, y_position, 'RANS',
557          ha='center', va='top', fontsize=14, fontweight='bold',
558          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
559 plt.text((9000-5000)/2+5000, y_position, 'Flush',
560          ha='center', va='top', fontsize=14, fontweight='bold',
561          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
562
563 plt.text((13000-9000)/2+9000, y_position, 'Opening',
564          ha='center', va='top', fontsize=14, fontweight='bold',
565          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
566
567 plt.text((17000-13000)/2+13000, y_position, 'Hold',
568          ha='center', va='top', fontsize=14, fontweight='bold',
569          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
570
571 plt.text((21000-17000)/2+17000, y_position, 'Closing',
572          ha='center', va='top', fontsize=14, fontweight='bold',
573          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
574
575 plt.text(23000, y_position, 'Final',
576          ha='center', va='top', fontsize=14, fontweight='bold',
577          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
578
579
580 ax.set_xlabel("Iteration Number", fontsize=12)
581 ax.set_ylabel("Normalized Cl", fontsize=12)
582 ax.set_title("Normalized Cl for the Complete Simulation - DRS_40_60",
    ↪ fontsize=14, fontweight='bold')
583 # ax.legend(loc='best', framealpha=0.9, fontsize=10)
584 ax.grid(True, alpha=0.3)
```

```
585 ax.set_ylim([None, 1.2])
586 plt.margins(x=0)
587 # plt.tight_layout()
588 plt.show()
589
590 # %% FILTER COMPARISON PLOTS (kept at bottom for visualization)
591
592 # Normalized drag - 2x2 grid comparison
593 fig, axes = plt.subplots(2, 2, figsize=(16, 12))
594
595 # Window sizes to compare
596 ma_windows = [30, 200]  # Two different moving average windows
597 sg_windows = [31, 201]  # Two different Savitzky-Golay windows (must be
    ↪  odd)
598
599 # Moving Average - Window 1
600 ax = axes[0, 0]
601 ax.plot(df_cleaned["Physical Time (s)"], df_percent["Drag - Total (NC)
    ↪  Monitor (N)"],
602         color="gray", linewidth=1, alpha=0.3, label="Original")
603 df_temp_ma1 = apply_moving_average(df_cleaned["Drag - Total (NC) Monitor
    ↪  (N)"], ma_windows[0])
604 ax.plot(df_cleaned["Physical Time (s)"],
605         (df_temp_ma1 / df_des_opening.iloc[0]["Drag - Total (NC) Monitor
            ↪  (N)"] - 1) * 100,
606         color="blue", linewidth=2, label=f"MA (window={ma_windows[0]})")
607
608 ax.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
609 ax.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
610 ax.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
611 ax.axvline(x=time_final_start, color='blue', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
612
613 ax.set_xlabel("Physical Time (s)", fontsize=11)
614 ax.set_ylabel("Drag (%)", fontsize=11)
615 ax.set_title(f"Moving Average - Window {ma_windows[0]}", fontsize=12,
    ↪  fontweight='bold')
616 ax.legend(loc='best', fontsize=9)
617 ax.grid(True, alpha=0.3)
618 ax.margins(x=0)
619
620 # Moving Average - Window 2
621 ax = axes[0, 1]
622 ax.plot(df_cleaned["Physical Time (s)"], df_percent["Drag - Total (NC)
    ↪  Monitor (N)"],
623         color="gray", linewidth=1, alpha=0.3, label="Original")
```

```
624 df_temp_ma2 = apply_moving_average(df_cleaned["Drag - Total (NC) Monitor
    ↪  (N)"], ma_windows[1])
625 ax.plot(df_cleaned["Physical Time (s)"],
626         (df_temp_ma2 / df_des_opening.iloc[0]["Drag - Total (NC) Monitor
            ↪  (N)"] - 1) * 100,
627         color="blue", linewidth=2, label=f"MA (window={ma_windows[1]})")
628
629 ax.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
630 ax.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
631 ax.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
632 ax.axvline(x=time_final_start, color='blue', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
633
634 ax.set_xlabel("Physical Time (s)", fontsize=11)
635 ax.set_ylabel("Drag (%)", fontsize=11)
636 ax.set_title(f"Moving Average - Window {ma_windows[1]}", fontsize=12,
    ↪  fontweight='bold')
637 ax.legend(loc='best', fontsize=9)
638 ax.grid(True, alpha=0.3)
639 ax.margins(x=0)
640
641 # Savitzky-Golay - Window 1
642 ax = axes[1, 0]
643 ax.plot(df_cleaned["Physical Time (s)"], df_percent["Drag - Total (NC)
    ↪  Monitor (N)"],
644         color="gray", linewidth=1, alpha=0.3, label="Original")
645 df_temp_sg1 = apply_savgol(df_cleaned["Drag - Total (NC) Monitor (N)"],
    ↪  sg_windows[0], 3)
646 ax.plot(df_cleaned["Physical Time (s)"],
647         (df_temp_sg1 / df_des_opening.iloc[0]["Drag - Total (NC) Monitor
            ↪  (N)"] - 1) * 100,
648         color="red", linewidth=2, label=f"Savgol
            ↪  (window={sg_windows[0]})")
649
650 ax.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
651 ax.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
652 ax.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
653 ax.axvline(x=time_final_start, color='blue', linestyle='--',
    ↪  linewidth=1.5, alpha=0.5)
654
655 ax.set_xlabel("Physical Time (s)", fontsize=11)
656 ax.set_ylabel("Drag (%)", fontsize=11)
657 ax.set_title(f"Savitzky-Golay - Window {sg_windows[0]}", fontsize=12,
    ↪  fontweight='bold')
```

```
ax.legend(loc='best', fontsize=9)
ax.grid(True, alpha=0.3)
ax.margins(x=0)

# Savitzky-Golay - Window 2
ax = axes[1, 1]
ax.plot(df_cleaned["Physical Time (s)"], df_percent["Drag - Total (NC) Monitor (N)"],
        color="gray", linewidth=1, alpha=0.3, label="Original")
df_temp_sg2 = apply_savgol(df_cleaned["Drag - Total (NC) Monitor (N)"], sg_windows[1], 3)
ax.plot(df_cleaned["Physical Time (s)"],
        (df_temp_sg2 / df_des_opening.iloc[0]["Drag - Total (NC) Monitor (N)"] - 1) * 100,
        color="red", linewidth=2, label=f"Savgol (window={sg_windows[1]})")

ax.axvline(x=time_opening_start, color='green', linestyle='--', linewidth=1.5, alpha=0.5)
ax.axvline(x=time_no_movement_start, color='violet', linestyle='--', linewidth=1.5, alpha=0.5)
ax.axvline(x=time_closing_start, color='orange', linestyle='--', linewidth=1.5, alpha=0.5)
ax.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=1.5, alpha=0.5)

ax.set_xlabel("Physical Time (s)", fontsize=11)
ax.set_ylabel("Drag (%)", fontsize=11)
ax.set_title(f"Savitzky-Golay - Window {sg_windows[1]}", fontsize=12, fontweight='bold')
ax.legend(loc='best', fontsize=9)
ax.grid(True, alpha=0.3)
ax.margins(x=0)

plt.suptitle("D Percentage - Filter Comparison", fontsize=14, fontweight='bold', y=0.995)
plt.tight_layout()
plt.show()

# %% DATA FRAMES FOR TRANSIENT ANALYSIS
# Filter data for Opening and Hold phases only from the already filtered df_cleaned
df_opening_hold = df_cleaned[
    (df_cleaned["Iteration"] >= 9001) & (df_cleaned["Iteration"] <= 17000)
].reset_index(drop=True)

df_close_final = df_cleaned[
    (df_cleaned["Iteration"] >= 17001) & (df_cleaned["Iteration"] <= 25000)
].reset_index(drop=True)
```

```
696
697 # %% SETTLING TIME ANALYSIS - DRAG (Opening & Hold)
698 settling_time, final_val, overshoot, overshoot_time =
    ↪ settling_time_analysis(
699     df=df_opening_hold,
700     time_col="Physical Time (s)",
701     value_col="Cl Monitor",
702     filtered_col="dummy",
703     settling_tolerance=0.01,
704     phase_name="Cl",
705     normalize=True,
706     reference_value=df_des_opening.iloc[0]["Cl Monitor"],
707     savgol_window=201,
708     savgol_poly=3,
709     time_opening_start=time_opening_start,
710     time_no_movement_start=time_no_movement_start,
711     time_closing_start=time_closing_start
712 )
713
714 if settling_time is not None:
715     print(f"\nSummary:")
716     print(f"Settling time: {settling_time:.4f} s")
717     print(f"Overshoot: {overshoot:.2f}%")
718     print(f"Overshoot time: {overshoot_time:.4f} s")
719     print(f"Final value: {final_val:.3f}")
720 else:
721     print(f"\nSummary:")
722     print(f"Signal does not settle within ś0.01 band")
723     print(f"Overshoot: {overshoot:.2f}%")
724     print(f"Overshoot time: {overshoot_time:.4f} s")
725     print(f"Final value: {final_val:.3f}")
726
727
728 # %% SETTLING TIME ANALYSIS - DRAG (Closing & Final)
729 settling_time_close, final_val_close, overshoot_close,
    ↪ overshoot_time_close = settling_time_analysis(
730     df=df_close_final,
731     time_col="Physical Time (s)",
732     value_col="Tire_Force % Rearwards Monitor",
733     filtered_col="dummy",
734     settling_tolerance=1,
735     phase_name="Tire_Force % Rearwards",
736     normalize=False,
737     reference_value=df_des_opening.iloc[0]["Tire_Force % Rearwards
        ↪ Monitor"],
738     savgol_window=201,
739     savgol_poly=3,
740     time_opening_start=time_closing_start,
741     time_no_movement_start=time_final_start,
742     time_closing_start=None
```

```
)

if settling_time_close is not None:
    print(f"\nClosing Phase Summary:")
    print(f"Settling time: {settling_time_close:.4f} s")
    print(f"Overshoot: {overshoot_close:.2f}%")
    print(f"Overshoot time: {overshoot_time_close:.4f} s")
    print(f"Final value: {final_val_close:.3f}")
else:
    print(f"\nClosing Phase Summary:")
    print(f"Signal does not settle within ś1.0 band")
    print(f"Overshoot: {overshoot_close:.2f}%")
    print(f"Overshoot time: {overshoot_time_close:.4f} s")
    print(f"Final value: {final_val_close:.3f}")

# %% FILTERED PLOTS - CORRECTED NORMALIZATION
#Normalized - FILTERED (CORRECTED REFERENCE)
# Get the reference value from the FILTERED data at the start of Opening (iteration 9001)
opening_first_idx = df_cleaned[df_cleaned['Iteration'] >= 9001].index[0]

# Create filtered percentage dataframe
df_percent_filtered = pd.DataFrame()
df_percent_filtered["Physical Time (s)"] = df_cleaned["Physical Time (s)"]

for col in drag_columns:
    filtered_col = f'{col}_SG_corrected' if col == 'Drag - Front Wing (NC) Monitor (N)' else f'{col}_SG'
    # Use the FILTERED value at the start of Opening as reference
    total_drag_reference = df_cleaned.loc[opening_first_idx, 'Drag - Total (NC) Monitor (N)_SG']
    df_percent_filtered[col] = ((df_cleaned[filtered_col] - df_cleaned.loc[opening_first_idx, filtered_col]) / total_drag_reference) * 100

fig, ax = plt.subplots(figsize=(14, 8))

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#bcbd22']

for i, col in enumerate(drag_columns):
    label = col.replace(' Monitor (N)', '').replace('Drag - ', '').replace(' (NC)', '')
    ax.plot(df_percent_filtered["Physical Time (s)"], df_percent_filtered[col],
            color=colors[i],
            linewidth=2.5,
            label=label,
            alpha=0.85)

```

```
785 plt.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
786 plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
787 plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
788 plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    ↪ alpha=0.7)
789
790 y_position = plt.ylim()[1] * 0.9
791
792 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
793          ha='center', va='top', fontsize=12, fontweight='bold',
794          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
795
796 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪ 'Opening',
797          ha='center', va='top', fontsize=12, fontweight='bold',
798          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
799
800 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪ 'Hold',
801          ha='center', va='top', fontsize=12, fontweight='bold',
802          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
803
804 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
805          ha='center', va='top', fontsize=12, fontweight='bold',
806          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
807
808 time_end = df_cleaned['Physical Time (s)'].iloc[-1]
809 plt.text((time_final_start + time_end)/2, y_position, 'Final',
810          ha='center', va='top', fontsize=12, fontweight='bold',
811          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
812
813 ax.set_title("Per Component Drag Contribution to Total Cd vs Physical Time
    ↪ (s) - Filtered - DRS_40_60", fontsize=16, fontweight='bold')
814 ax.set_ylabel("Cd / Cd_total (%)", fontsize=14)
815 ax.set_xlabel("Physical Time (s)", fontsize=14)
816
817 ax.yaxis.set_major_locator(plt.MultipleLocator(1))
818 ax.grid(True, alpha=0.3)
819
820 ax.legend(loc='best', framealpha=0.9, fontsize=14)
821 plt.tight_layout()
822 plt.margins(x=0)
823 plt.show()
824
825 #Normalized - FILTERED (Without Wheels and Front Wing components) -
    ↪ CORRECTED
826 drag_columns_filtered = [
```

```
827     'Drag - Body and Diffuser (NC) Monitor (N)',
828     'Drag - Rear Wing (NC) Monitor (N)',
829     'Drag - Side Wing (NC) Monitor (N)'
830 ]
831 
832 # Create filtered percentage dataframe
833 df_percent_filtered_nowheels = pd.DataFrame()
834 df_percent_filtered_nowheels["Physical Time (s)"] = df_cleaned["Physical
    ↪ Time (s)"]
835 
836 for col in drag_columns_filtered:
837     filtered_col = f'{col}_SG'
838     total_drag_reference = df_cleaned.loc[opening_first_idx, 'Drag - Total
        ↪ (NC) Monitor (N)_SG']
839     df_percent_filtered_nowheels[col] = ((df_cleaned[filtered_col] -
        ↪ df_cleaned.loc[opening_first_idx, filtered_col]) /
        ↪ total_drag_reference) * 100
840 
841 fig, ax = plt.subplots(figsize=(14, 8))
842 
843 colors = ['#1f77b4', '#8c564b', '#e377c2']
844 
845 for i, col in enumerate(drag_columns_filtered):
846     label = col.replace(' Monitor (N)', '').replace('Drag - ',
        ↪ '').replace(' (NC)', '')
847     ax.plot(df_percent_filtered_nowheels["Physical Time (s)"],
        ↪ df_percent_filtered_nowheels[col],
848             color=colors[i],
849             linewidth=2.5,
850             label=label,
851             alpha=0.85)
852 
853 plt.axvline(x=time_opening_start, color='green', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
854 plt.axvline(x=time_no_movement_start, color='violet', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
855 plt.axvline(x=time_closing_start, color='orange', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
856 plt.axvline(x=time_final_start, color='blue', linestyle='--', linewidth=2,
    ↪ alpha=0.7)
857 
858 y_position = plt.ylim()[1] * 0.9
859 
860 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
861          ha='center', va='top', fontsize=12, fontweight='bold',
862          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
863 
864 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪ 'Opening',
865          ha='center', va='top', fontsize=12, fontweight='bold',
```

```
866             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
867
868 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪  'Hold',
869             ha='center', va='top', fontsize=12, fontweight='bold',
870             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
871
872 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
873             ha='center', va='top', fontsize=12, fontweight='bold',
874             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
875
876 time_end = df_cleaned['Physical Time (s)'].iloc[-1]
877 plt.text((time_final_start + time_end)/2, y_position, 'Final',
878             ha='center', va='top', fontsize=12, fontweight='bold',
879             bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
880
881 ax.set_xlabel("Physical Time (s)", fontsize=14)
882 ax.set_ylabel("Cd / Cd_total (%)", fontsize=14)
883 ax.set_title("Per Component Drag Contribution to Total Cd vs Physical Time
    ↪  (s) - Main Aerodynamic Components - Filtered - DRS_40_60",
    ↪  fontsize=16, fontweight='bold')
884
885 ax.yaxis.set_major_locator(plt.MultipleLocator(1))
886 ax.grid(True, alpha=0.3)
887
888 ax.legend(loc='best', framealpha=0.9, fontsize=14)
889 plt.tight_layout()
890 plt.margins(x=0)
891 plt.show()
892
893 # # %% SAVE DATA FOR COMPARISON CODE
894 # Save all the key data you might want to compare
895 np.savez('DRS_40_60_data.npz',
896             # Time and iteration data
897             time=df_cleaned["Physical Time (s)"].values,
898             iteration=df_cleaned["Iteration"].values,
899             daoa=df_cleaned["DAoA"].values,
900
901             # Aerodynamic coefficients (raw)
902             cd=df_cleaned["Cd Monitor"].values,
903             cl=df_cleaned["Cl Monitor"].values,
904
905             # Total forces (raw)
906             total_drag=df_cleaned["Drag - Total (NC) Monitor (N)"].values,
907             total_downforce=df_cleaned["Downforce - Total Monitor"].values,
908
909             # Component drag (filtered with Savitzky-Golay)
910             body_drag_sg=df_cleaned["Drag - Body and Diffuser (NC) Monitor
                ↪  (N)_SG"].values,
```

```
911         front_wing_drag_sg=df_cleaned["Drag - Front Wing (NC) Monitor
            ↪  (N)_SG_corrected"].values,
912         front_wing_flaps_sg=df_cleaned["Drag - Front Wing Flaps (NC)
            ↪  Monitor (N)_SG"].values,
913         rear_wing_drag_sg=df_cleaned["Drag - Rear Wing (NC) Monitor
            ↪  (N)_SG"].values,
914         rear_wing_flap1_sg=df_cleaned["Drag - Rear Wing Flap 1 (NC)
            ↪  Monitor (N)_SG"].values,
915         rear_wing_flap2_sg=df_cleaned["Drag - Rear Wing Flap 2 (NC)
            ↪  Monitor (N)_SG"].values,
916         side_wing_drag_sg=df_cleaned["Drag - Side Wing (NC) Monitor
            ↪  (N)_SG"].values,
917         wheels_drag_sg=df_cleaned["Drag - Wheels Monitor (N)_SG"].values,
918         total_drag_sg=df_cleaned["Drag - Total (NC) Monitor
            ↪  (N)_SG"].values,
919
920         # Component drag (filtered with Moving Average)
921         body_drag_ma=df_cleaned["Drag - Body and Diffuser (NC) Monitor
            ↪  (N)_MA"].values,
922         front_wing_drag_ma=df_cleaned["Drag - Front Wing (NC) Monitor
            ↪  (N)_MA"].values,
923         rear_wing_drag_ma=df_cleaned["Drag - Rear Wing (NC) Monitor
            ↪  (N)_MA"].values,
924         side_wing_drag_ma=df_cleaned["Drag - Side Wing (NC) Monitor
            ↪  (N)_MA"].values,
925         wheels_drag_ma=df_cleaned["Drag - Wheels Monitor (N)_MA"].values,
926         total_drag_ma=df_cleaned["Drag - Total (NC) Monitor
            ↪  (N)_MA"].values,
927
928         # Balance
929         tire_force=df_cleaned["Tire_Force % Rearwards Monitor"].values,
930
931         # Normalized data (entire simulation including RANS)
932         cd_normalized=df_normalized["Cd Monitor"].values,
933         cl_normalized=df_normalized["Cl Monitor"].values,
934         iteration_normalized=df_normalized["Iteration"].values,
935
936         # Percentage change data (from df_percent)
937         body_drag_percent=df_percent["Drag - Body and Diffuser (NC)
            ↪  Monitor (N)"].values,
938         front_wing_drag_percent=df_percent["Drag - Front Wing (NC)
            ↪  Monitor (N)"].values,
939         rear_wing_drag_percent=df_percent["Drag - Rear Wing (NC) Monitor
            ↪  (N)"].values,
940         side_wing_drag_percent=df_percent["Drag - Side Wing (NC) Monitor
            ↪  (N)"].values,
941         wheels_drag_percent=df_percent["Drag - Wheels Monitor
            ↪  (N)"].values,
942         cd_percent=df_percent["Cd Monitor"].values,
943         cl_percent=df_percent["Cl Monitor"].values,
```

```
944             cd_percent_sg=df_percent["Cd Monitor_SG"].values,
945             cl_percent_sg=df_percent["Cl Monitor_SG"].values,
946
947             # Filtered percentage data (from df_percent_filtered)
948             body_drag_percent_filtered=df_percent_filtered["Drag - Body and
                ↪  Diffuser (NC) Monitor (N)"].values,
949             front_wing_drag_percent_filtered=df_percent_filtered["Drag -
                ↪  Front Wing (NC) Monitor (N)"].values,
950             rear_wing_drag_percent_filtered=df_percent_filtered["Drag - Rear
                ↪  Wing (NC) Monitor (N)"].values,
951             side_wing_drag_percent_filtered=df_percent_filtered["Drag - Side
                ↪  Wing (NC) Monitor (N)"].values,
952             wheels_drag_percent_filtered=df_percent_filtered["Drag - Wheels
                ↪  Monitor (N)"].values,
953             time_filtered=df_percent_filtered["Physical Time (s)"].values,
954
955             # Curve fit data
956             t_fit=t,
957             aoa_fit=aoa,
958             cop_fit=y_fit_t_cop,
959             cl_fit=y_fit_t_Cl,
960             cd_fit=y_fit_t_Cd,
961
962             # Phase transition times (for vertical lines)
963             time_flush_start=time_flush_start,
964             time_opening_start=time_opening_start,
965             time_no_movement_start=time_no_movement_start,
966             time_closing_start=time_closing_start,
967             time_final_start=time_final_start
968 )
969
970 print("\n Data saved to DRS_40_60_data.npz")
971 print(f"  - Raw data: Cd, Cl, component drags")
972 print(f"  - Filtered data: Savitzky-Golay and Moving Average")
973 print(f"  - Normalized data: Cd/Cl normalized to last Flush value")
974 print(f"  - Percentage data: Raw and filtered percentage changes")
975 print(f"  - Curve fits: CoP, Cl, Cd vs AoA")
976
977 # %% DRAG REDUCTION AND DOWNFORCE LOSS CALCULATION
978 # ============================================================
979 # Drag Reduction = (CD_closed - CD_open) / CD_closed * 100%
980 # Downforce Loss = (|CL_closed| - |CL_open|) / |CL_closed| * 100%
981 # CD_open  = mean over Hold phase (iterations 17001-21000)
982 # CD_closed = mean over Final phase (iterations 21001-25000)
983 # ============================================================
984
985 # Hold phase (open) - averaged over holding phase
986 df_hold = df_cleaned[
987     (df_cleaned["Iteration"] >= 13001) & (df_cleaned["Iteration"] <=
        ↪  17000)
```

```
]

# Final phase (closed) - averaged over final phase
df_final = df_cleaned[
    (df_cleaned["Iteration"] >= 21001) & (df_cleaned["Iteration"] <=
    ↪ 25000)
]

# Average Cd and Cl for each phase using filtered (SG) data
CD_open  = df_hold["Cd Monitor_SG"].mean()
CD_closed = df_final["Cd Monitor_SG"].mean()

CL_open  = df_hold["Cl Monitor_SG"].mean()
CL_closed = df_final["Cl Monitor_SG"].mean()

# Calculate drag reduction and downforce loss
drag_reduction = (CD_closed - CD_open) / CD_closed * 100
downforce_loss = (abs(CL_closed) - abs(CL_open)) / abs(CL_closed) * 100

print("\n" + "="*50)
print("DRS_40_60 - PERFORMANCE SUMMARY")
print("="*50)
print(f"  CD_open  (Hold phase mean):   {CD_open:.6f}")
print(f"  CD_closed (Final phase mean): {CD_closed:.6f}")
print(f"  CL_open  (Hold phase mean):   {CL_open:.6f}")
print(f"  CL_closed (Final phase mean): {CL_closed:.6f}")
print(f"\n  Drag Reduction:  {drag_reduction:.2f}%")
print(f"  Downforce Loss:  {downforce_loss:.2f}%")
print("="*50)
```

B.2 Transient Analysis Function

```
# settling_analysis.py
"""
Settling time analysis functions for transient CFD simulations.
Author: Panos Papakostas
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter

def settling_time_analysis(df, time_col, value_col, filtered_col,
                           settling_tolerance, phase_name="Opening and
                           ↪ Hold",
                           normalize=False, reference_value=None,
                           savgol_window=181, savgol_poly=3,
                           time_opening_start=None,
                           ↪ time_no_movement_start=None,
```

```
                              time_closing_start=None):
    """
    Generalized settling time analysis for any column with overshoot
    ↪ detection.

    Parameters:
    -----------
    df : DataFrame
        The dataframe containing the data
    time_col : str
        Column name for time data
    value_col : str
        Column name for original (unfiltered) values
    filtered_col : str
        Column name for filtered values (or will apply Savitzky-Golay if
        ↪ not pre-filtered)
    settling_tolerance : float
        Settling band tolerance (in normalized units if normalize=True)
    phase_name : str
        Name for the plot title
    normalize : bool
        If True, normalize values by dividing by reference_value
        ↪ (dimensionless)
    reference_value : float
        Reference value for normalization (required if normalize=True)
    savgol_window : int
        Window length for Savitzky-Golay filter (must be odd,
        ↪ default=181)
    savgol_poly : int
        Polynomial order for Savitzky-Golay filter (default=3)
    time_opening_start, time_no_movement_start, time_closing_start :
    ↪ float
        Time boundaries for phase annotations (optional)

    Returns:
    --------
    settling_time_relative : float or None
        The settling time relative to phase start, or None if signal
        ↪ doesn't settle
    final_value : float
        The calculated final steady-state value (normalized if
        ↪ normalize=True)
    overshoot_value : float
        The overshoot magnitude (as percentage of final value)
    overshoot_time_relative : float or None
        Time when overshoot occurs (relative to phase start)
    """

    # Extract data
    time = df[time_col].values
```

```
59      value_original = df[value_col].values
60
61      # Check if filtered column exists, otherwise apply Savitzky-Golay
62      if filtered_col in df.columns:
63          value_filtered = df[filtered_col].values
64      else:
65          # Apply Savitzky-Golay filter to original data
66          value_filtered = savgol_filter(value_original,
            ↪  window_length=savgol_window, polyorder=savgol_poly)
67
68      # Normalize if requested
69      if normalize:
70          if reference_value is None:
71              raise ValueError("reference_value must be provided when
                ↪  normalize=True")
72          value_original = value_original / reference_value
73          value_filtered = value_filtered / reference_value
74          ylabel = f"{value_col} (normalized)"
75      else:
76          ylabel = filtered_col if filtered_col in df.columns else value_col
77
78      # Define final value (average of last 10% of data)
79      final_10_percent_idx = int(len(df) * 0.9)
80      final_value = np.mean(value_filtered[final_10_percent_idx:])
81
82      print(f"Final steady-state value: {final_value:.3f}")
83
84      # Calculate overshoot
85      # Determine if signal is increasing or decreasing from start to final
86      initial_value = value_filtered[0]
87
88      if final_value > initial_value:
89          # Signal increases - look for overshoot above final value
90          overshoot_idx = np.argmax(value_filtered)
91          overshoot_abs = value_filtered[overshoot_idx]
92          overshoot_value = (overshoot_abs - final_value) / abs(final_value)
            ↪  * 100  # Percentage overshoot
93          overshoot_direction = "over"
94      else:
95          # Signal decreases - look for undershoot below final value
96          overshoot_idx = np.argmin(value_filtered)
97          overshoot_abs = value_filtered[overshoot_idx]
98          overshoot_value = (final_value - overshoot_abs) / abs(final_value)
            ↪  * 100  # Percentage undershoot
99          overshoot_direction = "under"
100
101     overshoot_time_absolute = time[overshoot_idx]
102
103     # Calculate relative overshoot time
104     if time_opening_start is not None:
```

```
105         overshoot_time_relative = overshoot_time_absolute -
            ↪ time_opening_start
106         print(f"Overshoot: {overshoot_value:.2f}%
            ↪ ({overshoot_direction}shoot)")
107         print(f"Overshoot time (absolute): {overshoot_time_absolute:.4f}
            ↪ s")
108         print(f"Overshoot time (relative): {overshoot_time_relative:.4f}
            ↪ s")
109     else:
110         overshoot_time_relative = overshoot_time_absolute
111         print(f"Overshoot: {overshoot_value:.2f}%
            ↪ ({overshoot_direction}shoot)")
112         print(f"Overshoot time: {overshoot_time_absolute:.4f} s")
113
114     # Define settling bands
115     upper_band = final_value + settling_tolerance
116     lower_band = final_value - settling_tolerance
117
118     # Find settling time (absolute)
119     settling_idx = None
120     for i in range(len(value_filtered)):
121         if all((value_filtered[i:] >= lower_band) & (value_filtered[i:] <=
            ↪ upper_band)):
122             settling_idx = i
123             break
124
125     # Calculate relative settling time
126     if settling_idx is not None:
127         settling_time_absolute = time[settling_idx]
128
129         # Calculate relative time (time since phase start)
130         if time_opening_start is not None:
131             settling_time_relative = settling_time_absolute -
                ↪ time_opening_start
132             print(f"Settling time (absolute): {settling_time_absolute:.4f}
                ↪ s")
133             print(f"Settling time (relative to phase start):
                ↪ {settling_time_relative:.4f} s")
134         else:
135             settling_time_relative = settling_time_absolute
136             print(f"Settling time: {settling_time_absolute:.4f} s (no
                ↪ phase start provided, returning absolute time)")
137     else:
138         settling_time_relative = None
139         settling_time_absolute = None
140         print(f"Signal does not settle within ś{settling_tolerance} band")
141
142     # Plot
143     fig, ax = plt.subplots(figsize=(14, 8))
144
```

```
145     # Plot original and filtered data
146     ax.plot(time, value_original,
147             color="gray", linewidth=1, alpha=0.5, label="Original")
148     ax.plot(time, value_filtered,
149             color="blue", linewidth=2.5, label=f"Filtered (Savgol,
            ↪ window={savgol_window})")
150
151     # Plot final value line
152     ax.axhline(y=final_value, color='green', linestyle=':', linewidth=2,
153                label=f'Final Value = {final_value:.3f}')
154
155     # Plot settling bands
156     ax.axhline(y=upper_band, color='red', linestyle='--', linewidth=1.5,
        ↪ alpha=0.7,
157                label=f'ś{settling_tolerance:.3f} Band')
158     ax.axhline(y=lower_band, color='red', linestyle='--', linewidth=1.5,
        ↪ alpha=0.7)
159     ax.fill_between(time, lower_band, upper_band, color='red', alpha=0.1)
160
161     # Plot overshoot point and line
162     ax.plot(overshoot_time_absolute, overshoot_abs, 'ro', markersize=10,
163             label=f'Overshoot = {overshoot_value:.2f}% at
            ↪ t={overshoot_time_relative:.4f}s' if time_opening_start
            ↪ else f'Overshoot = {overshoot_value:.2f}%',
164             zorder=5)
165     ax.axhline(y=overshoot_abs, color='darkred', linestyle=':',
        ↪ linewidth=1.5, alpha=0.5)
166
167     # Plot settling time (use absolute for the plot vertical line)
168     if settling_idx is not None:
169         label_text = f'Settling Time = {settling_time_relative:.4f} s' if
            ↪ time_opening_start is not None else f'Settling Time =
            ↪ {settling_time_absolute:.4f} s'
170         ax.axvline(x=settling_time_absolute, color='orange',
            ↪ linestyle='-', linewidth=2.5,
171                    label=label_text)
172
173     # Add vertical lines for phase transitions (if provided)
174     if time_no_movement_start is not None:
175         ax.axvline(x=time_no_movement_start, color='violet',
            ↪ linestyle='--',
176                    linewidth=2, alpha=0.7)
177
178     # Set axis labels and grid BEFORE annotations so ylim is correct
179     ax.set_xlabel("Physical Time (s)", fontsize=12)
180     ax.set_ylabel(ylabel, fontsize=12)
181     ax.set_title(f"{phase_name} - Transient Response Analysis -
        ↪ DRS_80_80", fontsize=14, fontweight='bold')
182     ax.legend(loc='best', framealpha=0.9, fontsize=10)
183     ax.grid(True, alpha=0.3)
```

```
    ax.margins(x=0)

    # Add phase annotations AFTER setting up the plot (so ylim is correct)
    if time_opening_start is not None and time_no_movement_start is not None:
        y_position = ax.get_ylim()[1] * 0.95  # 95% of y-axis height

        ax.text((time_opening_start + time_no_movement_start)/2, y_position,
                'Opening', ha='center', va='top', fontsize=14, fontweight='bold',
                bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

        if time_closing_start is not None:
            ax.text((time_no_movement_start + time_closing_start)/2, y_position,
                    'Hold', ha='center', va='top', fontsize=14, fontweight='bold',
                    bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

    plt.tight_layout()
    plt.show()

    return settling_time_relative, final_value, overshoot_value, overshoot_time_relative
```

B.3 Plot Comparison Postprocessing Script

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 25 11:33:37 2026

@author: panay
"""

# %% CODE FOR PLOT COMPARISON BETWEEN SIMULATIONS
import os
import numpy as np
import matplotlib.pyplot as plt

# Load both simulations
#Chose Simulation to Process
os.chdir(r'C:/Users/panay/Desktop/Thesis/40degs@60/CSV/')
data_60 = np.load('DRS_40_60_data.npz')
#Second Sim
os.chdir(r'C:/Users/panay/Desktop/Thesis/40degs@80/CSV/')
data_80 = np.load('DRS_40_80_data.npz')
```

```
20
21  """
22  Comparison Script for DRS Simulations
23  Compares 40deg/s @ 60kph vs 40deg/s @ 80kph
24  """
25
26  #
    ↪  ============================================================================================
27  # %% COMPARISON PLOTS
28  #
    ↪  ============================================================================================
29
30  # 1. Normalized Cd - Complete Simulation (vs Iteration)
31  #
    ↪  ============================================================================================
32  fig, ax = plt.subplots(figsize=(14, 8))
33
34  ax.plot(data_60['iteration_normalized'], data_60['cd_normalized'],
35          label='40ř/s @ 60kph', linewidth=2, color='blue')
36  ax.plot(data_80['iteration_normalized'], data_80['cd_normalized'],
37          label='40ř/s @ 80kph', linewidth=2, color='red')
38
39  # Phase lines
40  ax.axvline(x=5000, color='red', linestyle='--', linewidth=2, alpha=0.7)
41  ax.axvline(x=9000, color='green', linestyle='--', linewidth=2, alpha=0.7)
42  ax.axvline(x=13000, color='purple', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
43  ax.axvline(x=17000, color='orange', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
44  ax.axvline(x=21000, color='blue', linestyle='--', linewidth=2, alpha=0.7)
45
46  # Phase labels
47  y_position = ax.get_ylim()[1] * 0.1
48  plt.text(2500, y_position, 'RANS', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
49           bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
50  plt.text(7000, y_position, 'Flush', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
51           bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
52  plt.text(11000, y_position, 'Opening', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
53           bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
54  plt.text(15000, y_position, 'Hold', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
55           bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
56  plt.text(19000, y_position, 'Closing', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
57           bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
58  plt.text(23000, y_position, 'Final', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
```

```
59          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
60
61  ax.set_xlabel("Iteration Number", fontsize=14)
62  ax.set_ylabel("Normalized Cd", fontsize=14)
63  ax.set_title("Normalized Cd Comparison - Complete Simulation",
    ↪  fontsize=16, fontweight='bold')
64  ax.set_ylim([None, 1.2])
65  ax.legend(fontsize=12, loc='best')
66  ax.grid(True, alpha=0.3)
67  plt.margins(x=0)
68  plt.tight_layout()
69  plt.show()
70
71  # 2. Normalized Cl - Complete Simulation (vs Iteration)
72  #
    ↪  ============================================================================================
73  fig, ax = plt.subplots(figsize=(14, 8))
74
75  ax.plot(data_60['iteration_normalized'], data_60['cl_normalized'],
76          label='40ř/s @ 60kph', linewidth=2, color='blue')
77  ax.plot(data_80['iteration_normalized'], data_80['cl_normalized'],
78          label='40ř/s @ 80kph', linewidth=2, color='red')
79
80  # Phase lines
81  ax.axvline(x=5000, color='red', linestyle='--', linewidth=2, alpha=0.7)
82  ax.axvline(x=9000, color='green', linestyle='--', linewidth=2, alpha=0.7)
83  ax.axvline(x=13000, color='purple', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
84  ax.axvline(x=17000, color='orange', linestyle='--', linewidth=2,
    ↪  alpha=0.7)
85  ax.axvline(x=21000, color='blue', linestyle='--', linewidth=2, alpha=0.7)
86
87  # Phase labels
88  y_position = ax.get_ylim()[1] * 0.1
89  plt.text(2500, y_position, 'RANS', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
90          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
91  plt.text(7000, y_position, 'Flush', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
92          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
93  plt.text(11000, y_position, 'Opening', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
94          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
95  plt.text(15000, y_position, 'Hold', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
96          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
97  plt.text(19000, y_position, 'Closing', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
98          bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
```

```
99  plt.text(23000, y_position, 'Final', ha='center', va='top', fontsize=14,
    ↪  fontweight='bold',
100         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
101
102 ax.set_xlabel("Iteration Number", fontsize=16)
103 ax.set_ylabel("Normalized Cl", fontsize=16)
104 ax.set_title("Normalized Cl Comparison - Complete Simulation",
    ↪  fontsize=16, fontweight='bold')
105 ax.set_ylim([None, 1.2])
106 ax.legend(fontsize=12, loc='best')
107 ax.grid(True, alpha=0.3)
108 plt.margins(x=0)
109 plt.tight_layout()
110 plt.show()
111
112 # 3. Cd Percentage (vs Physical Time)
113 #
    ↪  ============================================================================
114 fig, ax = plt.subplots(figsize=(14, 8))
115
116 ax.plot(data_60['time'], data_60['cd_percent_sg'],
117         label='40ř/s @ 60kph', linewidth=2, color='blue')
118 ax.plot(data_80['time'], data_80['cd_percent_sg'],
119         label='40ř/s @ 80kph', linewidth=2, color='red')
120
121 # Phase lines (use 60kph times as reference)
122 ax.axvline(x=data_60['time_opening_start'], color='green', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
123 ax.axvline(x=data_60['time_no_movement_start'], color='violet',
    ↪  linestyle='--', linewidth=2, alpha=0.7)
124 ax.axvline(x=data_60['time_closing_start'], color='orange',
    ↪  linestyle='--', linewidth=2, alpha=0.7)
125 ax.axvline(x=data_60['time_final_start'], color='blue', linestyle='--',
    ↪  linewidth=2, alpha=0.7)
126
127 # Phase labels
128 y_position = ax.get_ylim()[1] * 0.9
129 time_flush_start = data_60['time_flush_start']
130 time_opening_start = data_60['time_opening_start']
131 time_no_movement_start = data_60['time_no_movement_start']
132 time_closing_start = data_60['time_closing_start']
133 time_final_start = data_60['time_final_start']
134 time_end = data_60['time'][-1]
135
136 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
137         ha='center', va='top', fontsize=12, fontweight='bold',
138         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
139 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪  'Opening',
140         ha='center', va='top', fontsize=12, fontweight='bold',
```

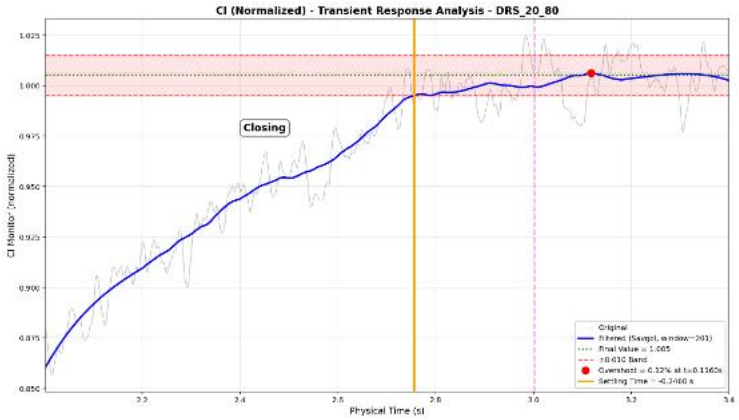
```
141         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
142 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪ 'Hold',
143         ha='center', va='top', fontsize=12, fontweight='bold',
144         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
145 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
146         ha='center', va='top', fontsize=12, fontweight='bold',
147         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
148 plt.text((time_final_start + time_end)/2, y_position, 'Final',
149         ha='center', va='top', fontsize=12, fontweight='bold',
150         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
151
152 ax.set_xlabel("Physical Time (s)", fontsize=16)
153 ax.set_ylabel("Cd (%)", fontsize=16)
154 ax.set_title("Cd Percentage Comparison - Filtered", fontsize=20,
    ↪ fontweight='bold')
155 ax.legend(fontsize=20, loc='best')
156 ax.grid(True, alpha=0.3)
157 plt.margins(x=0)
158 plt.tight_layout()
159 plt.show()
160
161 # 4. Cl Percentage (vs Physical Time)
162 #
    ↪ ============================================================================================
163 fig, ax = plt.subplots(figsize=(14, 8))
164
165 ax.plot(data_60['time'], data_60['cl_percent_sg'],
166         label='40ř/s @ 60kph', linewidth=2, color='blue')
167 ax.plot(data_80['time'], data_80['cl_percent_sg'],
168         label='40ř/s @ 80kph', linewidth=2, color='red')
169
170 # Phase lines
171 ax.axvline(x=data_60['time_opening_start'], color='green', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
172 ax.axvline(x=data_60['time_no_movement_start'], color='violet',
    ↪ linestyle='--', linewidth=2, alpha=0.7)
173 ax.axvline(x=data_60['time_closing_start'], color='orange',
    ↪ linestyle='--', linewidth=2, alpha=0.7)
174 ax.axvline(x=data_60['time_final_start'], color='blue', linestyle='--',
    ↪ linewidth=2, alpha=0.7)
175
176 # Phase labels
177 y_position = ax.get_ylim()[1] * 0.9
178
179 plt.text((time_flush_start + time_opening_start)/2, y_position, 'Flush',
180         ha='center', va='top', fontsize=12, fontweight='bold',
181         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
182 plt.text((time_opening_start + time_no_movement_start)/2, y_position,
    ↪ 'Opening',
```

```
183         ha='center', va='top', fontsize=12, fontweight='bold',
184         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
185 plt.text((time_no_movement_start + time_closing_start)/2, y_position,
    ↪  'Hold',
186         ha='center', va='top', fontsize=12, fontweight='bold',
187         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
188 plt.text((time_closing_start + time_final_start)/2, y_position, 'Closing',
189         ha='center', va='top', fontsize=12, fontweight='bold',
190         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
191 plt.text((time_final_start + time_end)/2, y_position, 'Final',
192         ha='center', va='top', fontsize=12, fontweight='bold',
193         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
194
195 ax.set_xlabel("Physical Time (s)", fontsize=16)
196 ax.set_ylabel("Cl (%)", fontsize=16)
197 ax.set_title("Cl Percentage Comparison - Filtered", fontsize=20,
    ↪  fontweight='bold')
198 ax.legend(fontsize=20, loc='best')
199 ax.grid(True, alpha=0.3)
200 plt.margins(x=0)
201 plt.tight_layout()
202 plt.show()
203
204 print("\n All comparison plots generated successfully!")
205
206 # %%Find minimum Cl for iterations > 10000 (60kph)
207 mask_60 = data_60['iteration'] > 10000
208 min_cl_60 = np.min(data_60['cl_percent_sg'][mask_60])
209 min_idx_60 = np.argmin(data_60['cl_percent_sg'][mask_60])
210 min_time_60 = data_60['time'][mask_60][min_idx_60]
211 min_iter_60 = data_60['iteration'][mask_60][min_idx_60]
212
213 print(f"\n60kph (iteration > 10000):")
214 print(f"  Minimum Cl: {min_cl_60:.2f}%")
215 print(f"  At iteration: {min_iter_60}")
216 print(f"  At time: {min_time_60:.4f} s")
217
218 # Same for 80kph
219 mask_80 = data_80['iteration'] > 10000
220 min_cl_80 = np.min(data_80['cl_percent_sg'][mask_80])
221 min_idx_80 = np.argmin(data_80['cl_percent_sg'][mask_80])
222 min_time_80 = data_80['time'][mask_80][min_idx_80]
223 min_iter_80 = data_80['iteration'][mask_80][min_idx_80]
224
225 print(f"\n80kph (iteration > 10000):")
226 print(f"  Minimum Cl: {min_cl_80:.2f}%")
227 print(f"  At iteration: {min_iter_80}")
228 print(f"  At time: {min_time_80:.4f} s")
229
230 # Find minimum Cd for iterations > 10000 (60kph)
```

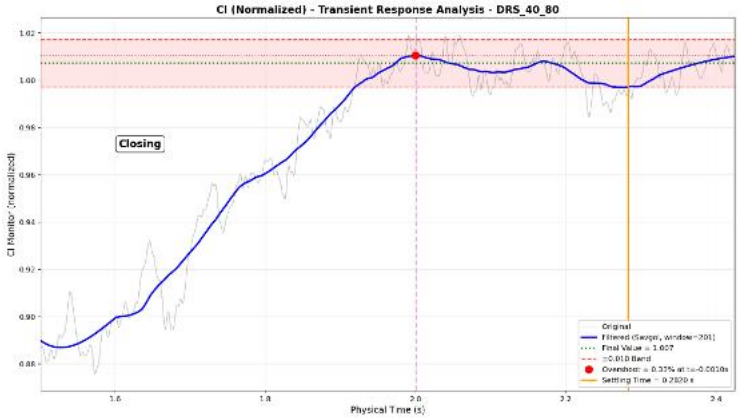
```
mask_60 = data_60['iteration'] > 10000
min_cd_60 = np.min(data_60['cd_percent_sg'][mask_60])
min_idx_60 = np.argmin(data_60['cd_percent_sg'][mask_60])
min_time_60 = data_60['time'][mask_60][min_idx_60]
min_iter_60 = data_60['iteration'][mask_60][min_idx_60]

print(f"\n60kph (iteration > 10000):")
print(f"  Minimum Cd: {min_cd_60:.2f}%")
print(f"  At iteration: {min_iter_60}")
print(f"  At time: {min_time_60:.4f} s")

# Same for 80kph
mask_80 = data_80['iteration'] > 10000
min_cd_80 = np.min(data_80['cd_percent_sg'][mask_80])
min_idx_80 = np.argmin(data_80['cd_percent_sg'][mask_80])
min_time_80 = data_80['time'][mask_80][min_idx_80]
min_iter_80 = data_80['iteration'][mask_80][min_idx_80]

print(f"\n80kph (iteration > 10000):")
print(f"  Minimum Cd: {min_cd_80:.2f}%")
print(f"  At iteration: {min_iter_80}")
print(f"  At time: {min_time_80:.4f} s")
```

C

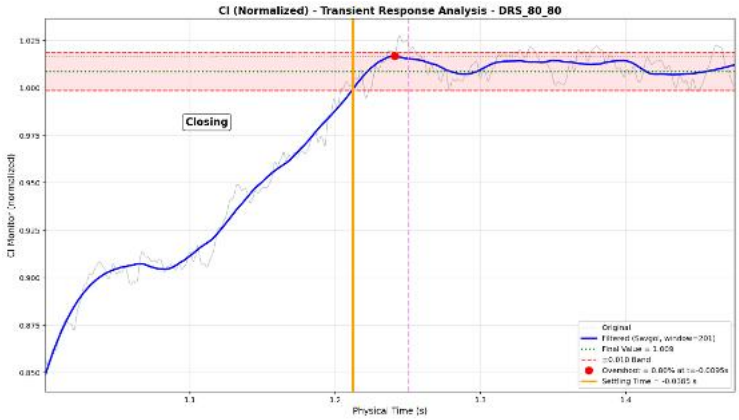
Appendix: Transient Response Curves



(a) $\omega = 20$ deg/s

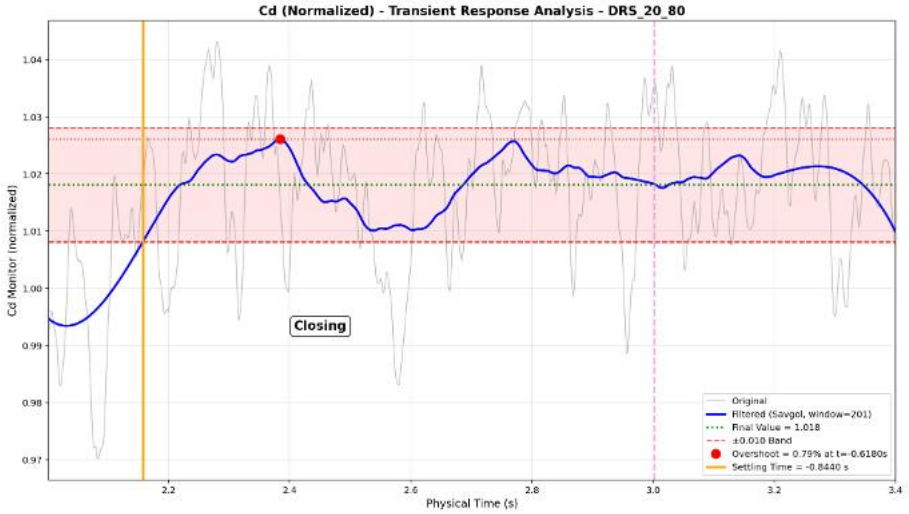


(b) $\omega = 40$ deg/s

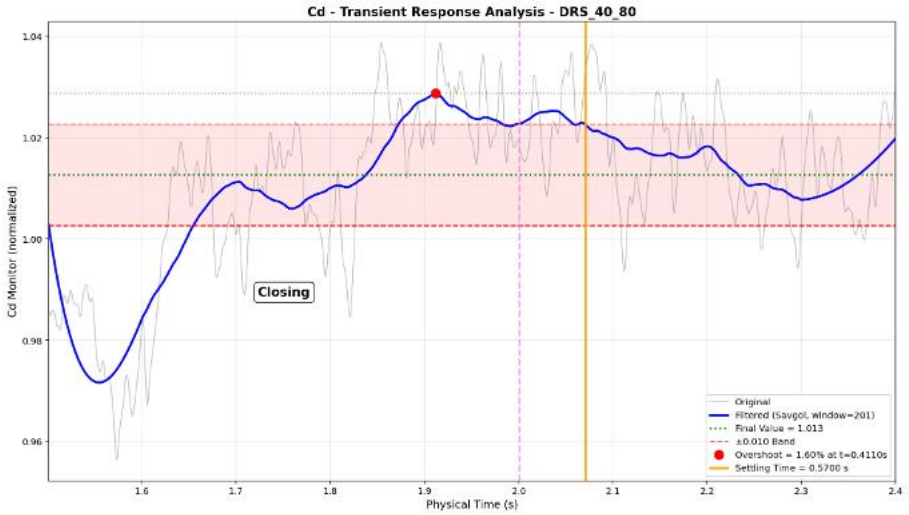


(c) $\omega = 80$ deg/s

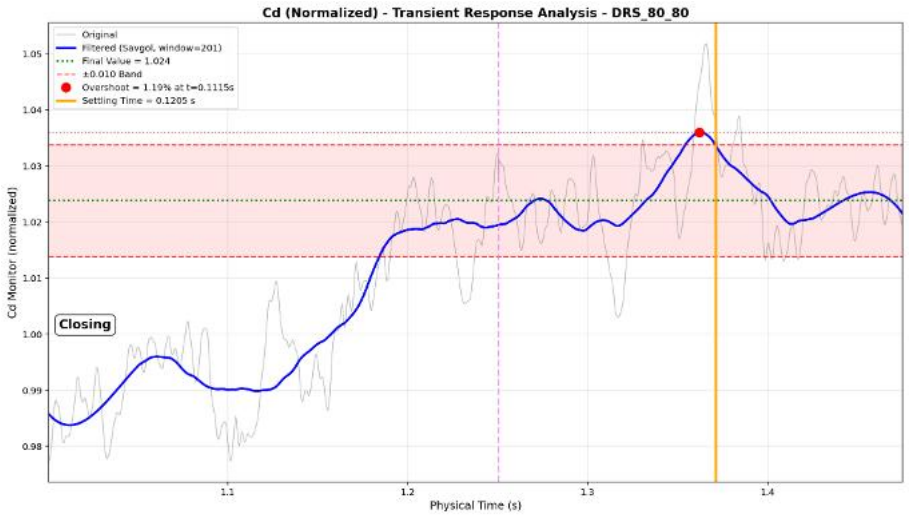
Figure C.1: Lift Coefficient (C_L) Transient Response During DRS Closing.



(a) $\omega = 20$ deg/s

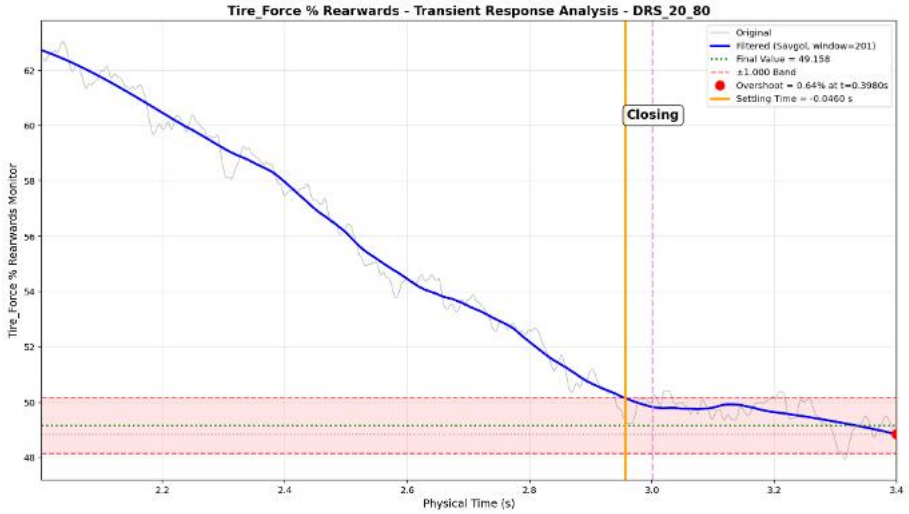


(b) $\omega = 40$ deg/s

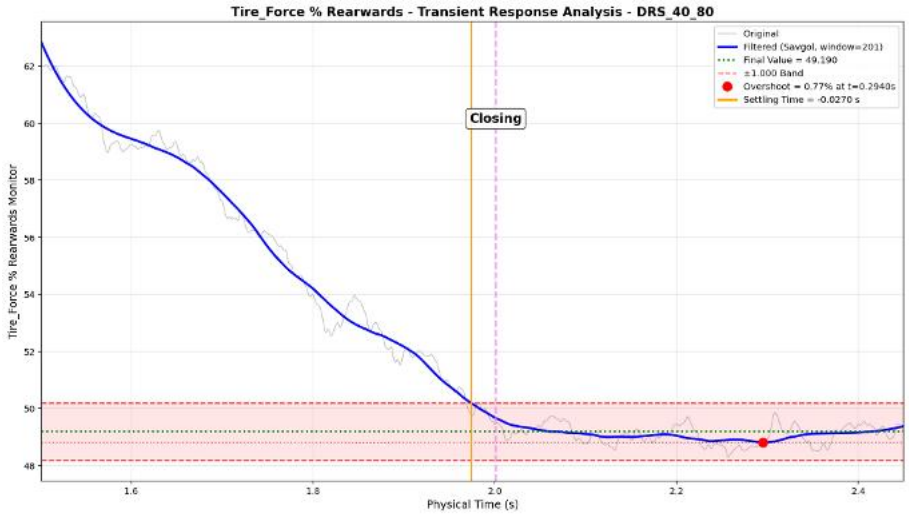


(c) $\omega = 80$ deg/s

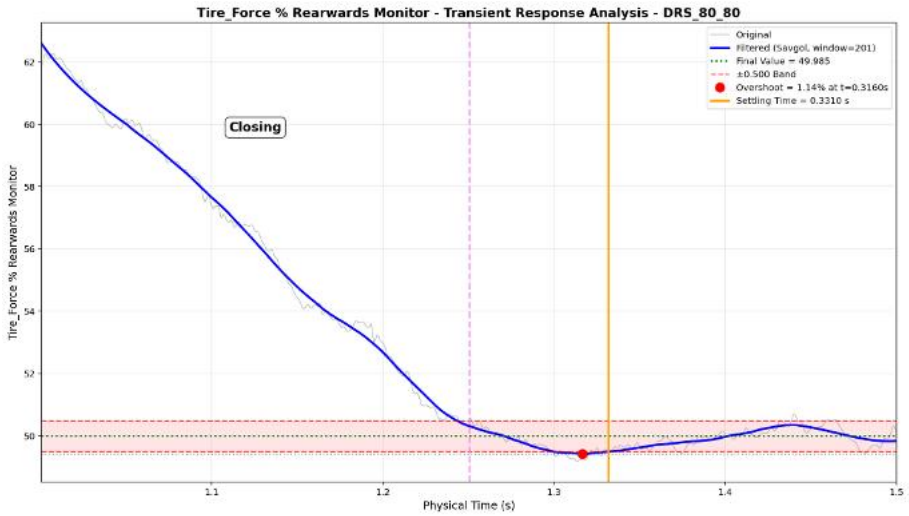
Figure C.2: Drag Coefficient (C_D) Transient Response During DRS Closing.



(a) $\omega = 20$ deg/s

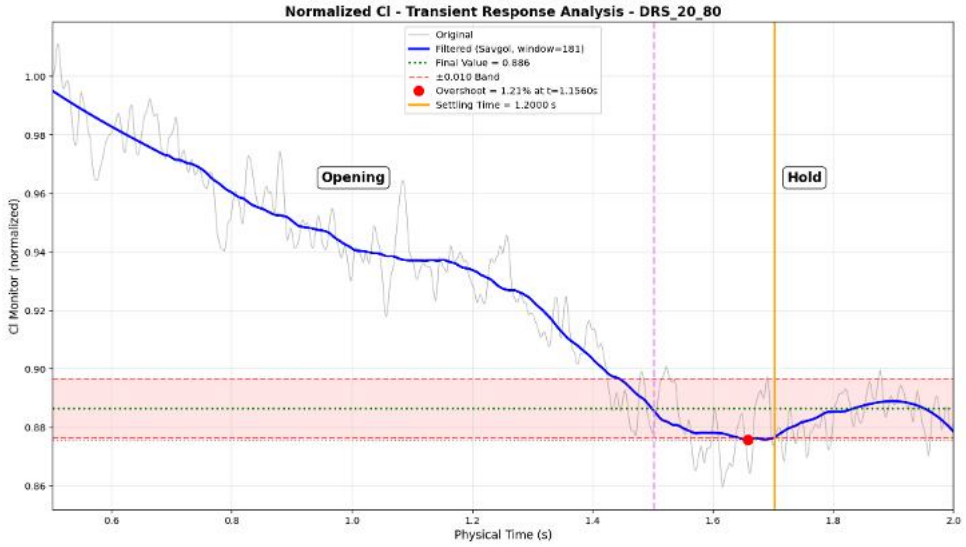


(b) $\omega = 40$ deg/s

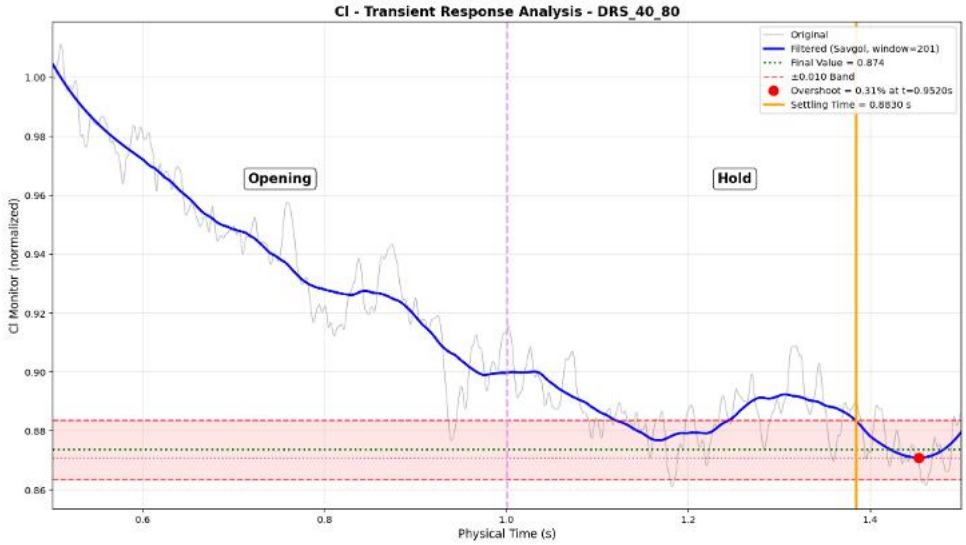


(c) $\omega = 80$ deg/s

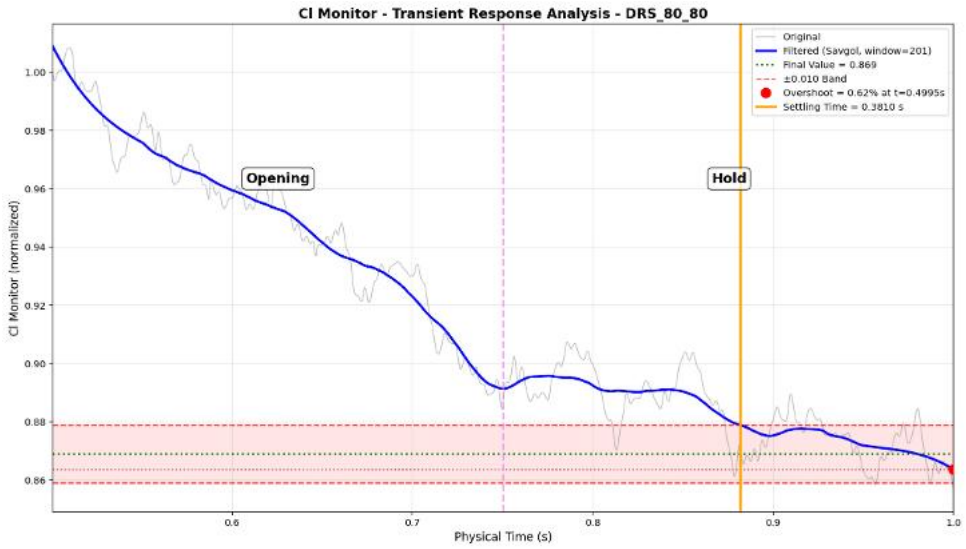
Figure C.3: Vehicle Balance (Tire Force % Rearwards) Transient Response During DRS Closing.



(a) $\omega = 20$ deg/s

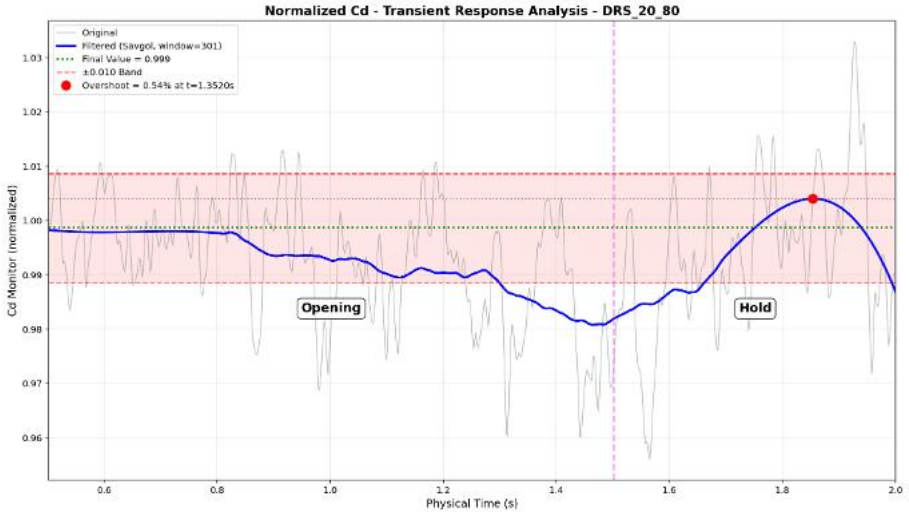


(b) $\omega = 40$ deg/s

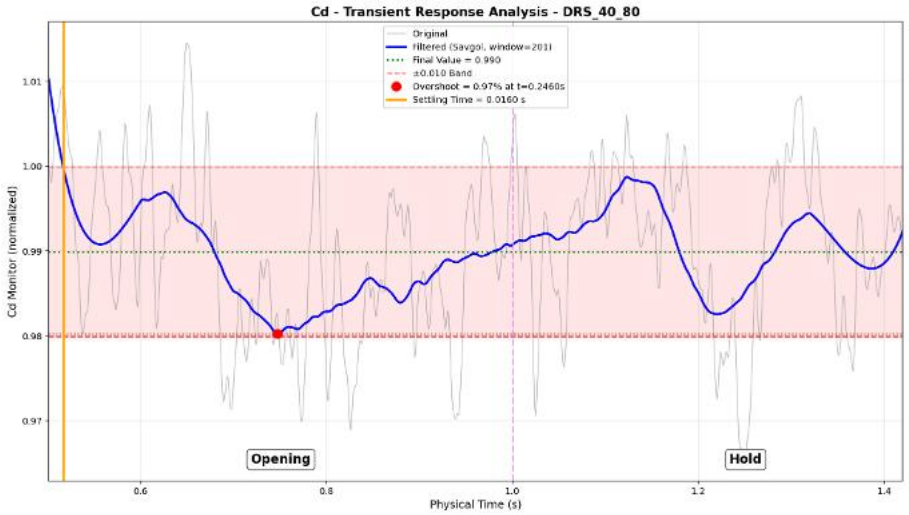


(c) $\omega = 80$ deg/s

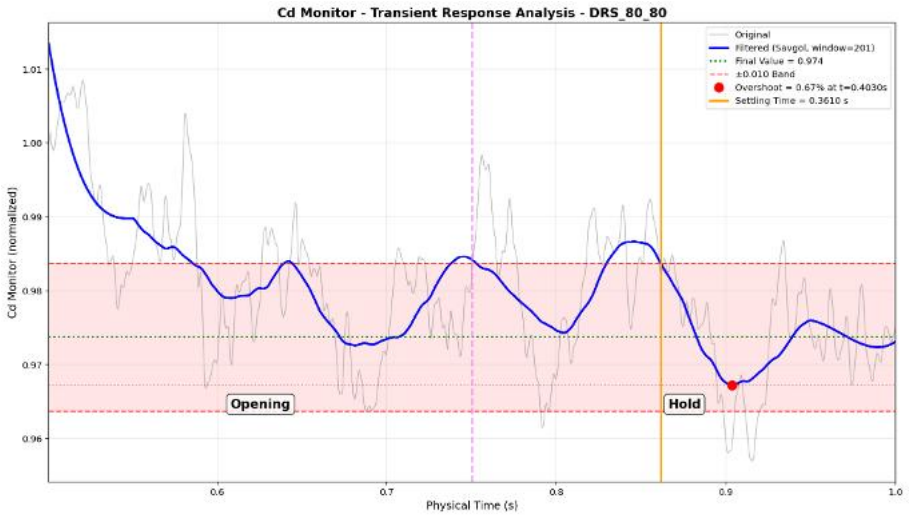
Figure C.4: Lift Coefficient (C_L) Transient Response During DRS Opening.



(a) $\omega = 20$ deg/s

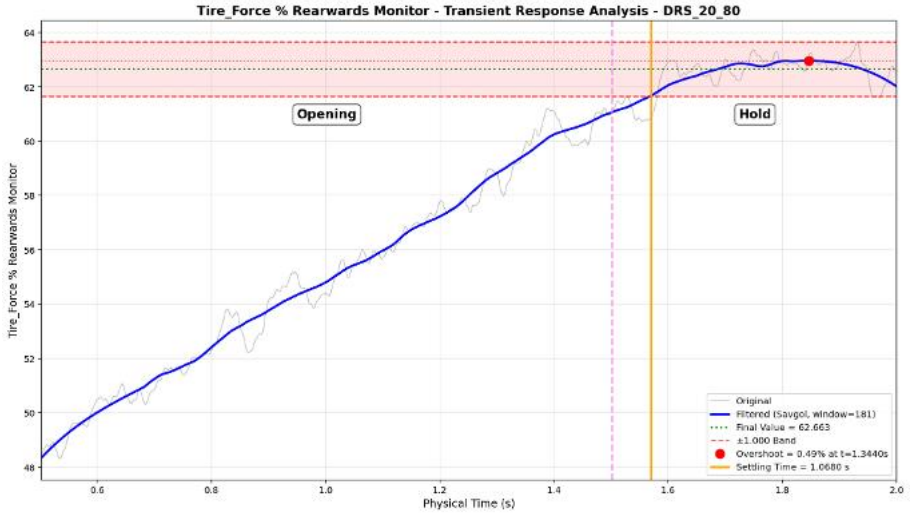


(b) $\omega = 40$ deg/s

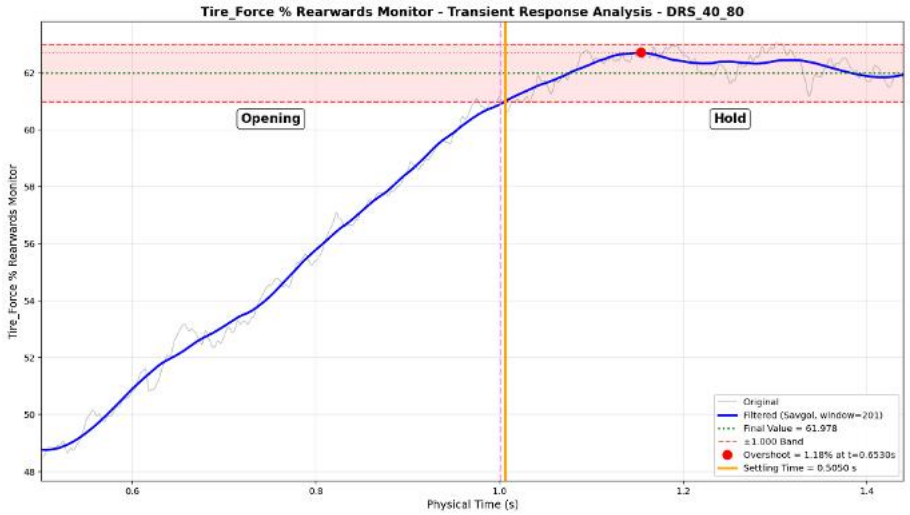


(c) $\omega = 80$ deg/s

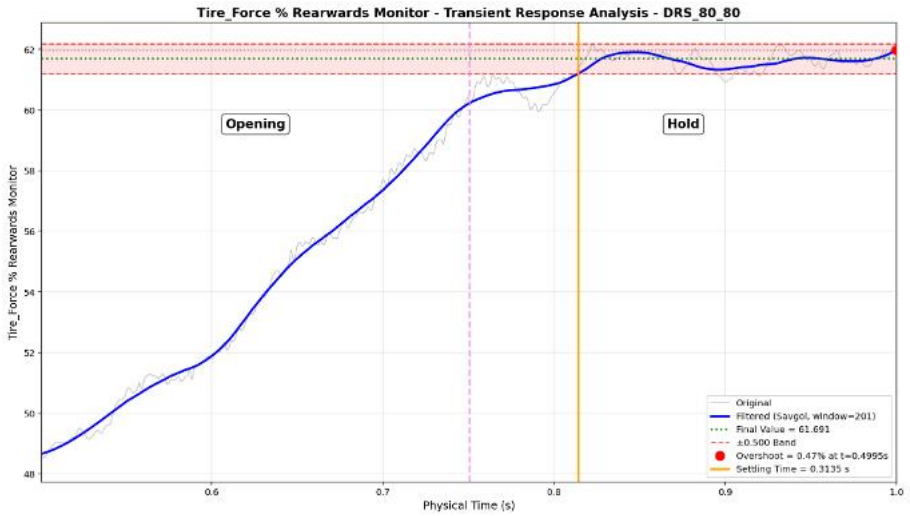
Figure C.5: Drag Coefficient (C_D) Transient Response During DRS Opening.



(a) $\omega = 20$ deg/s



(b) $\omega = 40$ deg/s



(c) $\omega = 80$ deg/s

Figure C.6: Vehicle Balance (Tire Force % Rearwards) Transient Response During DRS Opening.

D

Appendix: Fitted Curve Equations

The following fourth-degree polynomial equations represent the fitted curves for each configuration, where x is the independent variable.

DRS_20_60

$$f(x) = 4.10616038029183x^4 - 29.7536892454789x^3 + 66.0954375124276x^2 - 44.7677498258681x + 59.1550443869037 \quad \text{(D.1)}$$

DRS_20_80

$$f(x) = 2.4848313511455x^4 - 17.4495381980598x^3 + 34.5871328026575x^2 - 13.3912172643332x + 48.8929756489258 \quad \text{(D.2)}$$

DRS_40_80

$$f(x) = 28.5953570845277x^4 - 145.387664132213x^3 + 235.295814360643x^2 - 128.175382623598x + 70.5337443306521 \quad \text{(D.3)}$$

DRS_40_60

$$f(x) = 32.3038100932816x^4 - 166.62783077967x^3 + 278.393933836578x^2 - 163.918451080108x + 80.7617348000118 \quad \text{(D.4)}$$

DRS_80_80

$$f(x) = 502.10166791502x^4 - 1805.26795285109x^3 + 2263.21875693126x^2 - 1151.04650103067x + 252.975184596167 \quad \text{(D.5)}$$

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY