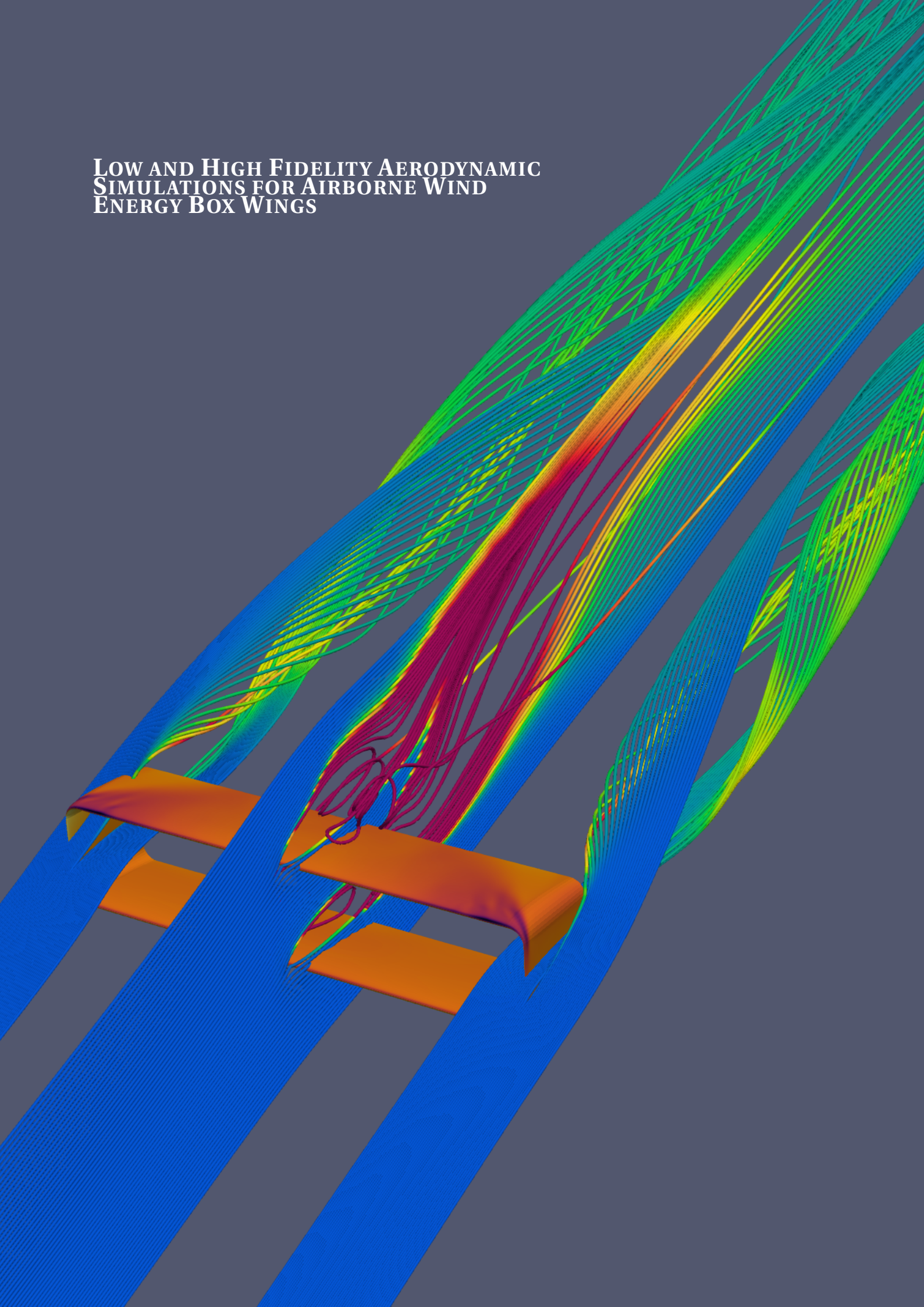
Low and High Fidelity Aerodynamic Simulations for Airborne Wind Energy Box Wings



LOW AND HIGH FIDELITY AERODYNAMIC SIMULATIONS FOR AIRBORNE WIND ENERGY BOX WINGS

by

Gabriel Eduardo Buendía Vela

to complete the subject AE4997: Educational Research Project
at the Delft University of Technology
to be defended publicly on Friday July 1, 2022 at 10:00 AM.

Student number:	5672546	
Project duration:	February, 2022-June, 2022	
Thesis committee:	Dr.-Ing. R. Schmehl	Tu Delft, chair and daily supervisor
	Dr. A. H van Zuijlen	Tu Delft, committee member
	Dr. D. J. N. Allaerts	Tu Delft, committee member
	PhD candidate D. Eijkelhof	Tu Delft, daily co-supervisor

PREFACE

This work has been a rewarding experience for me in the fields of aerodynamics and automation. I have gained much more experience in both fields than I initially had and I am leaving a tool for future generations that will help with the research on aerodynamics and wind energy. I believe that my research could serve as the basis for faster exploration of AWE concepts, saving computational time and engineering efforts thanks to task automation.

In the first place, I would like to express my sincere gratitude to my supervisor, Dr.-Ing Roland Schmehl, who gave me the opportunity to be part of this project and introduced me to the fascinating world of Airborne Wind Energy. I want to thank him for his patience, interest and the confidence placed in me to present my work in an international conference. My appreciation also extends to my co-supervisor PhD candidate Dylan Eijkelhof, who guided me when technical difficulties appeared and read each of this chapters of this work with dedication. Last but not least, I want to acknowledge with gratitude, the support and love of my parents, who encouraged me to move forward in difficult times and support me in all decisions I have taken.

ABSTRACT

Airborne wind energy systems convert the kinetic energy of wind into usable power. In general terms, this power is proportional to the ratio C_L^3/C_D^2 of aerodynamic coefficients. From a structural perspective, the thickness-to-chord ratio of conventional AWE wings needs to be high to withstand the high aerodynamic loads. The box-wing concept opens the possibility of exploring a broader range of airfoils since structural loads can be redistributed with reinforcements between the two wings. This study aims to develop an automatic process for constructing a finite volume CFD mesh from a parametrized box-wing geometry, which is generally the most time-demanding part of CFD analysis. These analyses provide an accurate estimate of the viscous drag acting on box wing designs. In addition, this study aims to define a criterion of equivalence between a box wing and a conventional wing, and obtain the reference design by optimization using panel methods for fast aerodynamic computations. The aerodynamic tools used for this study are a steady panel method (APAME) and Reynolds Averaged Navier-Stokes simulations using a k-ω SST turbulence model (OpenFOAM). The computational framework is ultimately suitable for aero-structural optimization of a box-wing because of the high degree of automation and the reduced number of design parameters.

CONTENTS

List of Figures

List of Tables

NOMENCLATURE

ACRONYMS

AEP	Annual Energy Production
AGL	Above Ground Level
APAME	Aircraft 3D Panel Method
API	Application Programming Interface
ARP	Aerospace Recommended Practices
AWE	Airborne Wind Energy
AWES	Airborne Wind Energy Systems
BW	Box Wing
CAD	Computer-Aided Design
CAPEX	Capital Expenditures
CFD	Computational Fluid Dynamics
CRF	Capital Recovery Factor
CS	Certification Specifications
CW	Conventional Wing
EASA	European Aviation Safety Agency
FDM	Finite Difference Method
FEM	Finite Element Method
FG-AWES	Fly-Gen Airborne Wind Energy System
FVM	Finite Volume Method
GAMG	Geometric Algebraic Multi Grid
GG-AWES	Ground-Gen Airborne Wind Energy System
ICC	Initial Capital Cost
IEA	International Energy Agency
LCOE	Levelized Cost of Energy
LEI	Leading Endge Inflatable
MoC	means of Compliance
NAA	National Aviation Authority
NACA	National Advisory Committee for Aeronautics
NZE	Net Zero Emissions
OMC	Operations and Maintenance Costs
RANS	Reynolds Averaged Navier-Stokes
RPAS	Remotely Piloted Aircraft Systems
SC	Special Conditions
SIMPLE	Semi-Inplicit Method for Pressure Linked Equations
SIMPLEC	Semi-Inplicit Method for Pressure Linked Equations Consistent
SST	Shear Stress Transport
STEPS	Stated Policies Scenario
TKE	Turbulent Kinetic Energy
TRL	Technological Readiness Level
UNCCTEC	United Nations Climate Change Technology Executive Committee
WPI	Wind Potential Index

LATIN SYMBOLS

A	Area of the wing skin	$[m^2]$
A_{BW}	Area of the box wing skin	$[m^2]$

A_{CW}	Area of the conventional wing skin	$[m^2]$
a_1	k-ω SST coefficient	$[-]$
B	Richardson extrapolation coefficient	$[-]$
b	Wingspan	$[m]$
b_{BW}	Wingspan box wing	$[m]$
b_{CW}	Wingspan conventional wing	$[m]$
C_{ϵ_1}	Standard k-ϵ coefficient	$[-]$
C_{ϵ_2}	Standard k-ϵ coefficient	$[-]$
C_μ	Standard k-ϵ coefficient	$[-]$
$C_{D,power}$	Non intrinsic drag forces coefficient	$[-]$
$C_{L,APAME}$	3D Lift coefficient from APAME	$[-]$
$C_{L,BW}$	3D Lift coefficient box wing	$[-]$
$C_{L,CW}$	3D Lift coefficient conventional wing	$[-]$
$C_{L,Experimental}$	3D Lift coefficient from experimental data	$[-]$
$C_{L\alpha}$	Lift coefficient slope	$[-]$
C_{L0}	Lift Coefficient at zero angle of attack	$[-]$
c_{r1}	Lower wing root chord	$[m]$
c_{r2}	Upper wing root chord	$[m]$
c_{t1}	Lower wing tip chord	$[m]$
c_{t2}	Upper wing tip chord	$[m]$
C_D	3D Drag coefficient	$[-]$
C_f	Skin friction coefficient	$[-]$
C_L	3D Lift coefficient	$[-]$
c_l	Local chord lower wing	$[m]$
C_p	Pressure coefficient	$[-]$
c_u	Local chord upper wing	$[m]$
D	Drag force	$[N]$
E	Specific total energy	$[Jkg^{-1}]$
$\vec{f}_e$	External specific forces	$[ms^{-2}]$
f	Variable of interest for Richardson extrapolation	$[-]$
H	Specific enthalpy	$[Jkg^{-1}]$
h	Box wing height	$[m]$
i	Aparent order of the numerical method	$[-]$
I	Integral difference in lift coefficient curves	$[-]$
$\bar{\bar{I}}$	Identity tensor	$[-]$
k	Turbulent kinetic energy	$[m^2s^{-2}]$
k_c	Thermal conductivity	$[Wm^{-1}K^{-1}]$
k_i	Initial turbulent kinetic energy	$[m^2s^{-2}]$
L	Lift force	$[N]$
l_c	Characteristic length	$[m]$
l_m	Turbulent length scale	$[m]$
M	Mach number	$[-]$
MAC	Mean aerodynamic chord	$[m]$
MAC_l	Mean aerodynamic chord lower wing	$[m]$
MAC_u	Mean aerodynamic chord upper wing	$[m]$
N_{af}	Number of points to discretize half airfoil	$[-]$
N_{bl}	Number of lower wing half spanwise sections	$[-]$
N_{bu}	Number of upper wing half spanwise sections	$[-]$
N_{bw}	Number of winglet sections	$[-]$
N_{it}	Number of iterations evaluated for Cauchy error	$[-]$
N_R	Number of points to discretize curved regions	$[-]$
N_{sim}	Number of iterations ran in CFD simulation	$[-]$
p	Pressure	$[Pa]$
p_i	Initial pressure	$[Pa]$
P	Useful power	$[W]$
Q_∞	Dynamic pressure	$[kgm^{-1}s^{-2}]$

q_H	Specific heat sources	$[Jm^{-3}s^{-1}]$
R	Cant radius	$[m]$
Re	Reynolds number	$[-]$
s	Box wing stagger	$[m]$
S	Wing surface	$[m^2]$
$S_{ref,BW}$	Reference wing area of the box wing	$[m^2]$
$S_{ref,CW}$	Reference wing area of the conventional wing	$[m^2]$
S_{ref}	Reference wing area (planform)	$[m^2]$
S_l	Reference area lower wing	$[m^2]$
S_u	Reference area upper wing	$[m^2]$
T	Temperature	$[K]$
t	Thickness of the wing skin	$[m]$
T_∞	Freestream temperature	$[K]$
T_I	Turbulent intensity	$[-]$
$\vec{u}$	Velocity	$[ms^{-1}]$
$\vec{U}_i$	Initial velocity	$[ms^{-1}]$
U	Velocity magnitude	$[ms^{-1}]$
U_∞	Freestream velocity magnitude	$[ms^{-1}]$
u_τ	Friction velocity	$[ms^{-1}]$
U_x	Velocity component parallel to the freestream	$[ms^{-1}]$
V	Wind speed	$[ms^{-1}]$
W	Wing weight	$[N]$
W_f	Specific external work	$[Jm^{-3}s^{-1}]$
x	Coordinate in the x axis of the farfield reference frame	$[m]$
x_{af}	Coordinate in the chordwise airfoil direction axis	$[m]$
y	Coordinate in the y axis of the farfield reference frame	$[m]$
y^+	Dimensionless wall distance	$[-]$
z	Coordinate in the z axis of the farfield reference frame	$[m]$
z_{af}	Coordinate in the axis connecting airfoil leading edges	$[m]$

GREEK SYMBOLS

α	Angle of attack	$[^\circ]$
α_f	Upper angle of attack bound	$[^\circ]$
α_I	Intersection angle of attack	$[^\circ]$
α_i	Lower angle of attack bound	$[^\circ]$
α_p	Pressure under relaxation factor	$[-]$
α_v	Velocity under relaxation factor	$[-]$
β	Angle of sideslip	$[^\circ]$
β'	k-ω coefficient	$[-]$
β^*	k-ω coefficient	$[-]$
β_1	k-ω SST coefficient	$[-]$
β_2	k-ω SST coefficient	$[-]$
δ	Kronecker delta	$[-]$
δ_F	First layer thickness	$[m]$
ϵ	Turbulent dissipation rate	$[m^2s^{-3}]$
ϵ_c	Cauchy convergence error	$[-]$
γ	k-ω coefficient	$[-]$
γ_1	k-ω SST coefficient	$[-]$
γ_2	k-ω SST coefficient	$[-]$
λ	Taper ratio	$[-]$
μ	Dynamic viscosity	$[kgm^{-1}s^{-1}]$
μ_∞	Freestream dynamic viscosity	$[kgm^{-1}s^{-1}]$
μ_D	Strength of doublet	$[m^2s^{-1}]$
ν	Kinematic viscosity	$[m^2s^{-1}]$

ν_∞	Freestream kinematic viscosity	$[m^2 s^{-1}]$
ν_t	Turbulent viscosity	$[m^2 s^{-1}]$
ω	Specific turbulence dissipation	$[s^{-1}]$
ω_i	Initial specific turbulent dissipation	$[s^{-1}]$
$\vec{\omega}$	Flow vorticity	$[s^{-1}]$
ω_n	Flow vorticity in the plane normal to x_{af}	$[s^{-1}]$
ω_x	Flow vorticity component parallel to the freestream	$[s^{-1}]$
ω_z	Flow vorticity component perpendicular to the freestream and the spanwise direction	$[s^{-1}]$
$\overline{\Omega}$	Mean flow rotation	$[s^{-1}]$
$\overline{\overline{\tau}}$	Shear stress tensor	$[Pa]$
ϕ	Velocity potential	$[m^2 s^{-1}]$
ρ	Density	$[kgm^3]$
ρ_∞	Freestream density	$[kgm^{-3}]$
ρ_i	Initial density	$[kgm^{-3}]$
ρ_s	Density of the wing skin	$[kgm^{-3}]$
σ	k-ω coefficient	$[-]$
σ^*	k-ω coefficient	$[-]$
σ_ϵ	Standard k-ϵ coefficient	$[-]$
σ_{ω_1}	k-ω SST coefficient	$[-]$
σ_{ω_2}	k-ω SST coefficient	$[-]$
σ_k	Standard k-ϵ coefficient	$[-]$
σ_{k1}	k-ω SST coefficient	$[-]$
σ_{k2}	k-ω SST coefficient	$[-]$
σ_{SS}	Strength of source or sink	$[m^2 s^{-1}]$
τ_w	Wall shear stress	$[Pa]$

1

INTRODUCTION

The International Energy Agency (IEA) issued a summary of the current state and forecasts of world energy and emissions [1]. In particular three main scenarios are studied in the forecasts: Net Zero Emissions by 2050 Scenario (NZE), Announced Pledges Scenario (APS) which takes into account the commitments made by different countries through the world and the Stated Policies Scenario (STEPS) which contains the current measures applied by the governments. According to this report, electricity was expected to increase beyond 1000 TWh in 2021, after the decline experienced due to the pandemic in 2020. Renewable energies became really important as a source of electricity generation in 2020, reaching a share of 28% and making the emissions fall a 3% in this sector. Figure 1.1 illustrates the change in global electricity generation and is seen how renewables were expected to become important in 2021 as well. In terms of CO_2 emissions they were expected

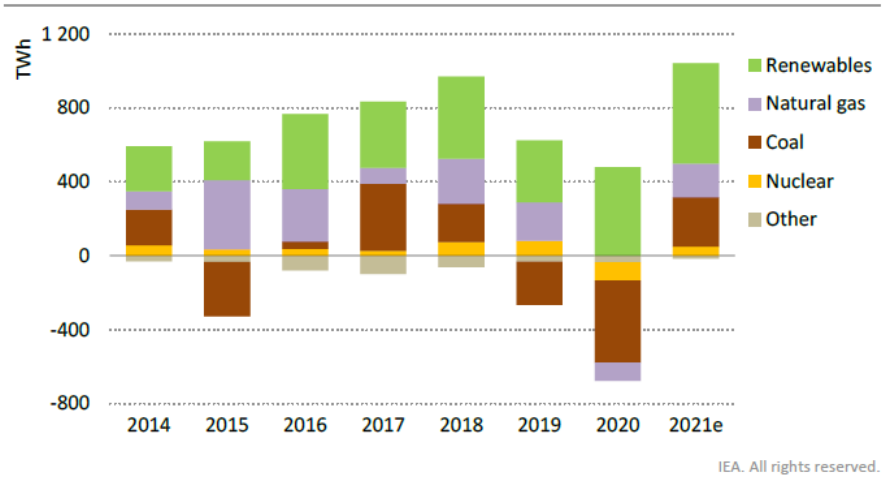


Figure 1.1: Change in global electricity generation [1].

to increase up to 1.2 billion tonnes, which means they will cover 2/3 of the reduction experienced because of the pandemic in 2020. Figure 1.2 illustrates the annual change in CO_2 emissions together with the energy source. This means that in order to reduce the fossil fuel emissions and usage, a higher technological development and usage of renewable energies must be set. In terms of the electricity supply, both wind and solar photo voltaic are expected to rise their share from around 10% in 2020 to 23%, 27% and 40% in the STEPS, APS and NZE forecasts for 2030, respectively. Figure 1.3 illustrates the importance of each energy source in the variation of electricity generation between 2020 and 2030.

Between 2023 and 2025 both offshore and onshore wind capacity additions are expected to increase and make a total average between 65 and 90 GW for the normal and accelerated case, respectively. However, wind projects face several challenges including social acceptance, permitting and policy uncertainties among others. Figure 1.4 illustrates the global additions between years 2020 and 2025. The reason why onshore capacity growth is faster lies on European and American policies for repowering and refurbishment of old wind tur-

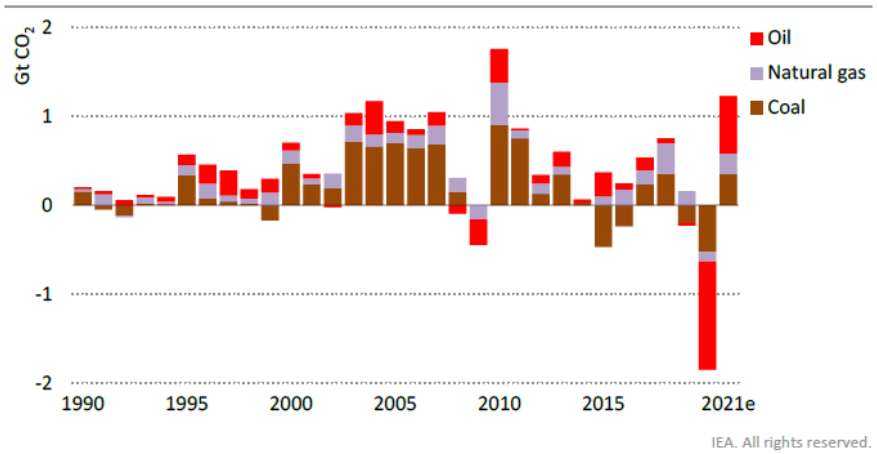


Figure 1.2: Annual change in CO_2 related to energy [1].

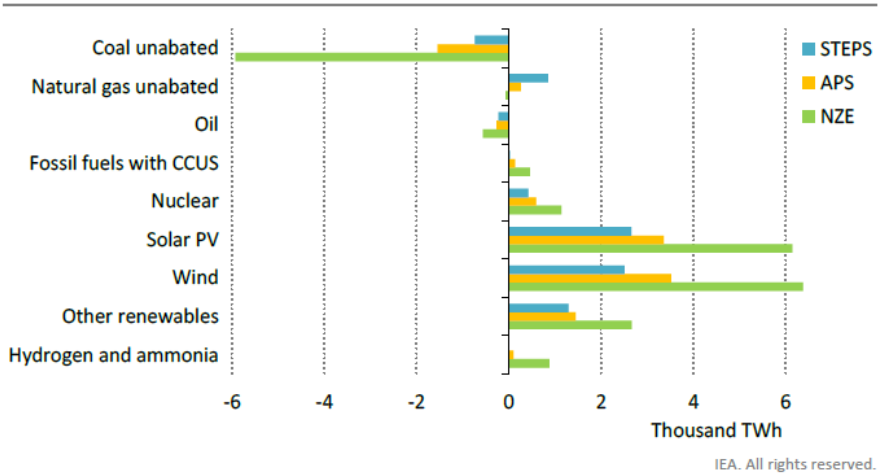


Figure 1.3: Change in electricity generation by source and scenario from 2020 to 2030 [1].

bines [2]. The study made by Jung & Schindler [3] intended to create a global Wind Potential Index (WPI) to identify the regions suitable for consistent wind farm settlement around the world. This was necessary since there were discrepancies found between places with high resource potential, meaning high energy yields and high geographical potential, as well as good accessibility for operation and construction. Examples of countries with high wind resource but poor accessibility include parts of Canada, Russia and some desserts. However, the WPI helped to identify regions like Argentina, USA or England where a good compromise between accessibility and energy yield was found.

Regarding conventional wind turbines, there is a continuous increase on the rotor diameter and the hub height aiming to reach higher wind power densities. In fact, nameplate capacity has increased from 1 MW (1998) to 7.58 MW (2016) for onshore wind turbines and from 1.5 MW (1998) to 8 MW (2015) for offshore wind turbines [4]. However, the issue of this growth is the saturation of key areas suitable for wind extraction. In addition, large-scale wind turbines for further power extraction will require technological innovations and significant cost reductions that will still need to solve for the power capacity per area maximization and aeroelastic instabilities [5]. Airborne Wind Energy (AWE) is a relatively new research area for wind energy extraction that can reach higher altitudes, requires less material for construction and could comply with the power demands conventional wind turbines have.

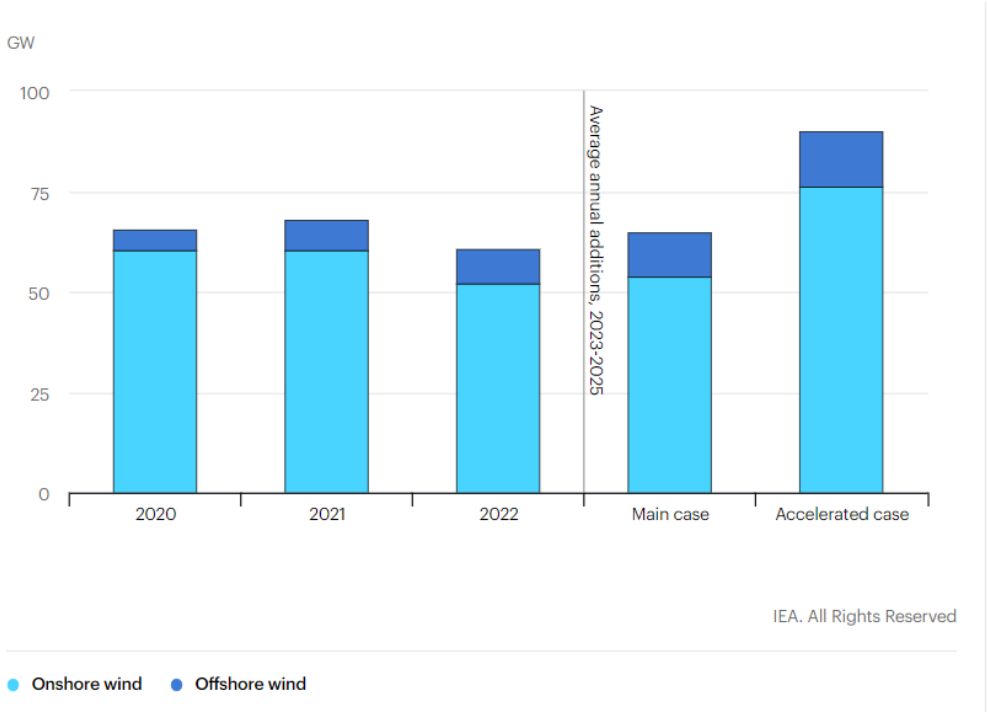


Figure 1.4: Global wind annual net capacity additions from 2020 to 2025 [2].

1.1. AWES

Airborne Wind Energy Systems (AWES) convert the kinetic energy of the wind into electrical energy. The kites used for power extraction operate in two different conditions: free flight or ground attached. The last condition, which is the most common AWE system, relies on a tether connecting the kite with a fixed or moving ground station [6]. Among AWE systems it is possible to distinguish between Ground-Gen (GG) and Fly-Gen (FG) systems. GG-AWES produce energy at the ground station by exploiting the work done by the traction force on an electrical generator. As previously mentioned the ground station can be either fixed or movable. FG-AWES produce energy on the kite itself by making use of propellers functioning as small wind turbines. This energy is transmitted to the ground via a rope, called a tether and they can be further classified into crosswind or non-crosswind regarding the way they produce energy [7]. Some examples of manufactured AWE systems are classified and shown in Figure 1.5.

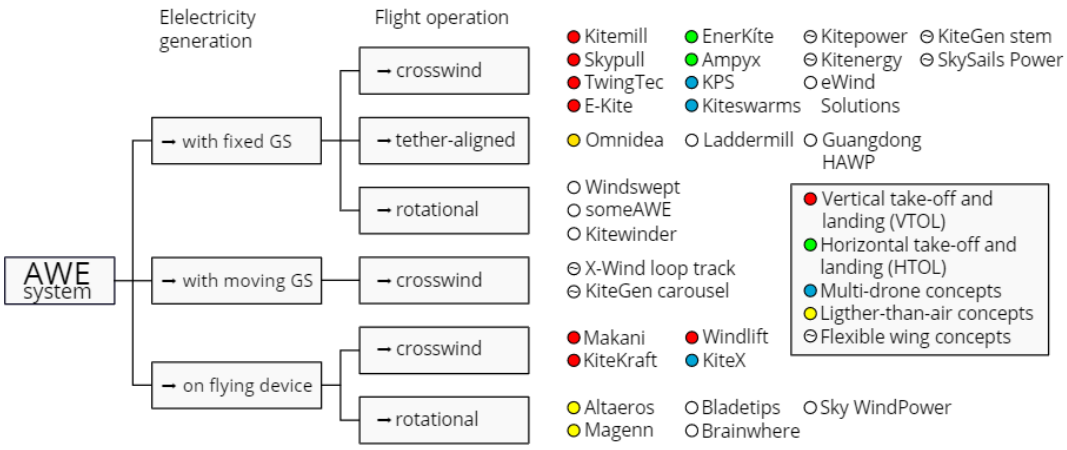


Figure 1.5: Classification of AWE systems [8].

The main advantages of AWES against conventional wind turbines are the lower use of material and thus capital costs, easier installation and construction, and higher energy yield due to the higher operational altitudes (above 200 m) [9]. If they are deployed offshore, then potential benefits in weight reduction and subsequently in CAPEX are expected since the loading case is different. In offshore wind turbines the main focus is on bending of the tower thus heavy ballast and marine structures are needed, whereas for tethered AWES the main concern is about tensile loads [10].

However, some important existing challenges need to be assessed, including long periods of continuous fully autonomous flight, lack of a representative demonstrator that can operate without constant supervision and limited empirical data, which means revenue or climate benefit predictions are limited as well [11].

1.2. MOTIVATION FOR THIS WORK

The key motivation features of this work can be summarized into two: saving engineering time and get reliable simulation data. The challenge of having reliable aerodynamic simulations in a reasonable time affects not only AWES, but several disciplines inside and outside the aerospace industry.

This work was strongly focused on reducing the computational time that the aerodynamic design process requires. This process includes the selection of an optimal geometry, the meshing and pre-processing for CFD simulations, the simulation monitoring and the final post-processing. The reduction is achieved by identifying and automating several tasks the user is not strictly required to work in.

In terms of AWES, the automated framework was validated and used to explore new designs such as the rigid box wing concept. A concept that promised aerodynamic advantages when compared to the conventional rigid wing designs.

1.3. OUTLINE

The structure of this work is presented as follows. Chapter 2 describes the box wing concept, together with the numerical and aerodynamic studies found in the literature. Brief sections regarding the socioeconomic impact and regulatory framework found in the literature are also included in this chapter. Chapter 3 is focused on the objectives and the methodology followed for the complete development of this work. Chapter 4 describes the governing equations of fluid dynamics and the numerical discretization for two different approaches. Chapter 5 aims to give a detailed description of the definition of the vehicle, the meshing process and the initial setup for both low fidelity and high fidelity aerodynamic simulations. Validation of the tools used is also presented in this chapter. Chapter 6 focuses on the post processing of the high fidelity simulations used for validation. Finally, a parametric study and optimization using the low fidelity aerodynamic tool is presented in Chapter 7.

2

LITERATURE REVIEW

In this chapter the box wing concept is explained in detail. Firstly, some insight on the monoplane kites used by the industry. Secondly, biplane performance is introduced followed by the introduction to box wings and the typical configurations for aviation. In the third place, some examples of box wings applied to the AWEs field are presented. Fourth, aerodynamic modelling approaches are introduced and the research gap is identified for this particular work. Finally, a brief socioeconomic impact analysis of this work is presented together with the regulatory framework that would be applicable.

2.1. MONOPLANES

Chapter 1 showed there is a large variety of kites that can be used for AWE systems. In particular, Cherubini et al. [7] distinguish up to seven different types of kites. Leading Edge Inflatable (LEI), ram-air kites, delta kites, gliders, swept rigid wings, semi-rigid wings and special design kites. Regardless of their nature they are mostly single wing and most companies are changing to rigid wing designs due to durability and performance issues [12]. Examples regarding the performance of rigid monoplane prototypes are Makani M600, which was capable of generating up to 600 kW using eight turbines on-board each having five propeller blades [13], and Ampyx Power AP-3, which was capable of generating up to 150 kw with no turbines on-board [14].

Figure 2.1: Examples of rigid monoplanes applied to AWE systems: M600 [13] (left) and AP-3 [14] (right)

2.2. BIPLANES

Since the beginning of the aviation biplanes became really popular, being the first flying machine invented in 1903 by the Wright Brothers. However, their popularity decayed in the mid 1930s since they were claimed to present inherent drag, which was not useful for military applications, but also a difficult aerodynamic design [15].

Improvements in the design, the materials and also in computational technology allowed to compare and state superior performance of biplanes against monoplanes in certain applications. They include excellent

low speed manoeuvrability, good load carrying ability and good short field landing and takeoff performance among others [16]. Although the biplane configuration achieved lower maximum lift coefficients, a drag reduction of 25% and a consequent aerodynamic efficiency increase of 31.2% compared to an equivalent monoplane was achieved experimentally in Olson & Selberg [17].

One of the fundamental equations in AWE systems is considered to be Loyd's formula, which describes the useful power for crosswind kite systems is:

$$P = \frac{2}{27}\rho S C_L \left(\frac{C_L}{C_D}\right)^2 V^3 \tag{2.1}$$

With C_L and C_D the lift and drag coefficients, ρ is the air density at the flying altitude, S is the wing surface and V the wind speed. With this in mind it is possible to state that biplane kites are a suitable option for AWEs since the power produced increases as the aerodynamic efficiency of the kite squared [18].

In addition, there are adapted versions of Loyd's formula [6][19] taking into account non intrinsic drag forces ($C_{D,power}$) such as on-board turbines or the tether connecting the kite to the ground station. The useful power for crosswind kite system is now:

$$P = \frac{2}{27}\rho S C_R \left(\frac{C_R}{C_D}\right)^2 V^3 \tag{2.2}$$

with:

$$C_R = C_L \sqrt{1 + \left(\frac{C_D + C_{D,power}}{C_L}\right)^2} \tag{2.3}$$

2.3. BOX WINGS

According to Prandtl [20] the best wing system in terms of induced drag was the box wing configuration which implies closing the biplane configuration on each of its sides as depicted in Fig. 2.2. Note that G denotes height and b wingspan.

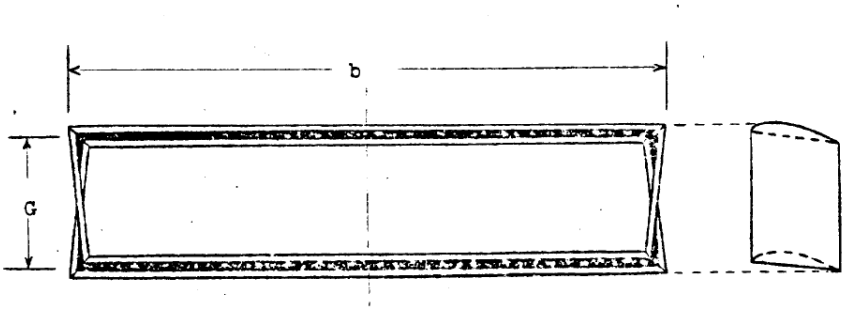


Figure 2.2: Best wing system according to Prandtl [20].

The fact that the aerodynamic characteristics of a biplane wing could be enhanced by transforming it into a box wing was checked both theoretically and experimentally in Gall & Smith [21]. Aerodynamic efficiency was shown to increase 6.4% with respect to the biplane case. Figures 2.3 and 2.4 show aerodynamic comparisons between different aircraft configurations, being the box-wing design in both cases the one leading to best performances.

2.4. BOX WINGS ON AWES

When talking about box wings almost all literature focuses on the Prandtl Plane concept applied to transport aircraft, this is why the aerodynamic approaches are discussed in the following section taking into account these references. However, there are a couple of airborne wind energy companies that have successfully manufactured their box wing kites.

Joby Energy Inc. focused on multi frame structure with airfoils. They included airborne wind turbine generators within the structure [23]. Different small scale prototypes (similar to Fig. 2.5) were tested between 2009 and 2010. Their goal was to build airborne wind turbines that would operate in the upper boundary layer (Earth) and the upper troposphere.

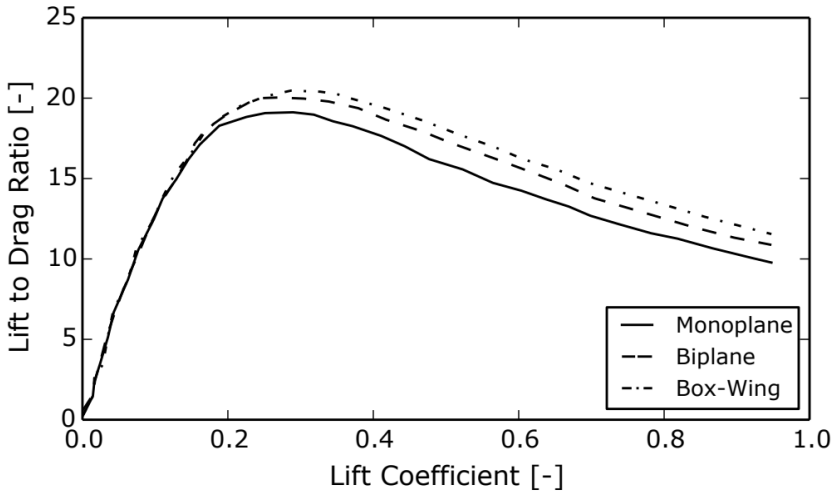


Figure 2.3: Aerodynamic efficiency comparison taken from Andrews [22] (Adapted from Gall & Smith [21]).

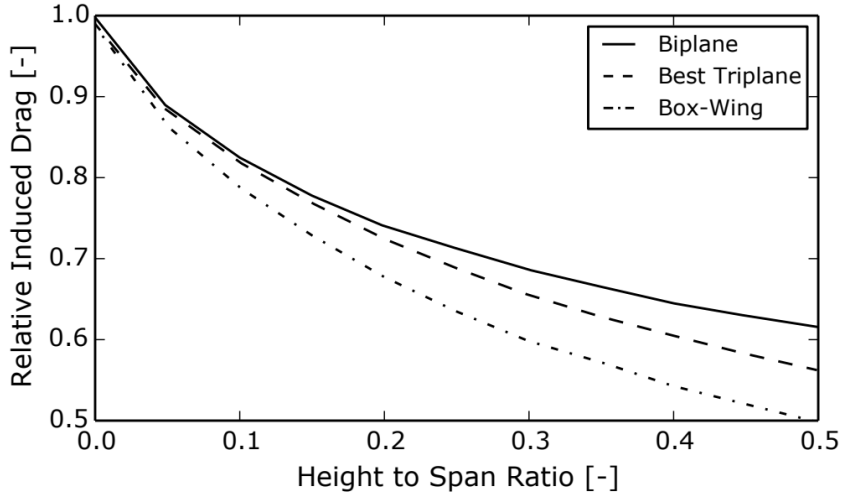


Figure 2.4: Induced drag relative to an elliptically loaded monoplane comparison taken from Andrews et al. [22](Adapted from Prandtl [20]).

Kitekraft relies on small tethered aircraft to extract wind energy. They have developed a box plane truss-like structure aircraft with aluminum extruded wings, carbon fiber rotor blades and multiple control surfaces. They claim benefits such as power density increase, optimum harvesting efficiency, good stability and weight reduction [24]. Figure 2.6 represents the 100kW product that KiteKraft would like to have ready by 2024. There has been successful prototype tests from the company that can already fly all flight phases [25].

2.5. BOX WINGS AERODYNAMICS

During the last decade there have been several studies considering box wings and tandem wings configurations. Unfortunately, most of these studies an aerodynamic approaches are focused on box wings for transport aircrafts. However, the state of art and the tools used for aerodynamic analysis are still relevant to this research.

Khan [26] conducted a parametric study is conducted to find the minimum induced drag. To study the aerodynamic effect, vortex lattice algorithms are used as well as Euler inviscid simulations. It is concluded that the most important parameter is the height to span ratio, since the higher it is, the lower the induced

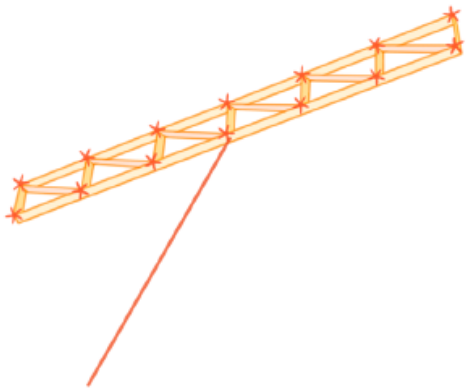


Figure 2.5: Kite designed by Joby Energy taken from Cherubini et al. [7].

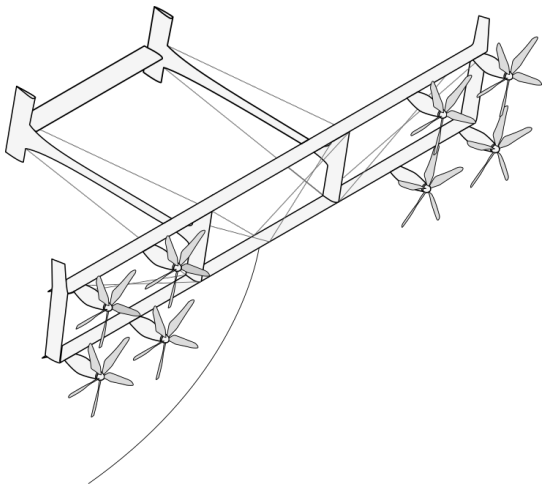


Figure 2.6: Kite design taken from KiteKraft [24].

drag becomes. Some improvement in induced drag is also achieved by varying the cant angles from the winglets (see Figure 2.7). Other parameters such as stagger, sweep and taper ratio are shown to have no effect if the proper span loading is maintained. Finally, this analysis does not include viscous effects, structural nor stability considerations but it is still giving some physical insight for box wings preliminary aerodynamic calculations.

Gagnon & Zingg [27] varied parameters such as the twist, sweep, section and center of mass for a transonic box-wing regional jet configuration using inviscid simulations. Five different optimizations are conducted starting with twist and section variation, and progressively adding more design variables and constraints. The final results are compared against normal regional jet configurations. The box-wing design is shown to be more efficient in terms of inviscid pressure drag and stability than the conventional wing design under this flight conditions (M = 0.78). However, no viscous drag study nor structural analysis are conducted in this paper. The box-wing configuration used in this analysis is depicted in Figure 2.8.

Andrews & Perez [28] compared the performance of a conventional and a box-wing aircraft. The parametric study used a vortex panel aerodynamic model including some fixed drag polar corrections for the airfoils used (NACA 23012). By tuning parameters such as the wing area distribution, the stagger and the height to span ratio (Figure 2.9) it is concluded that the box-wing design can reach higher aerodynamic efficiency than the conventional wing design. The fuel consumption is also analyzed in this paper but it is out of the scope of the present work. In the multidisciplinary analysis, a structural model is introduced for the box-wing: hexagonal wingbox with stress carrying booms at each vertex which are connected through shear carrying skins.

Bauer et al. [29] analyzed the performance of small scale monoplane and biplane kites with onboard wind turbines. CFD is used with a RANS solver and the k-ω turbulence model for the aerodynamics. Amongst all

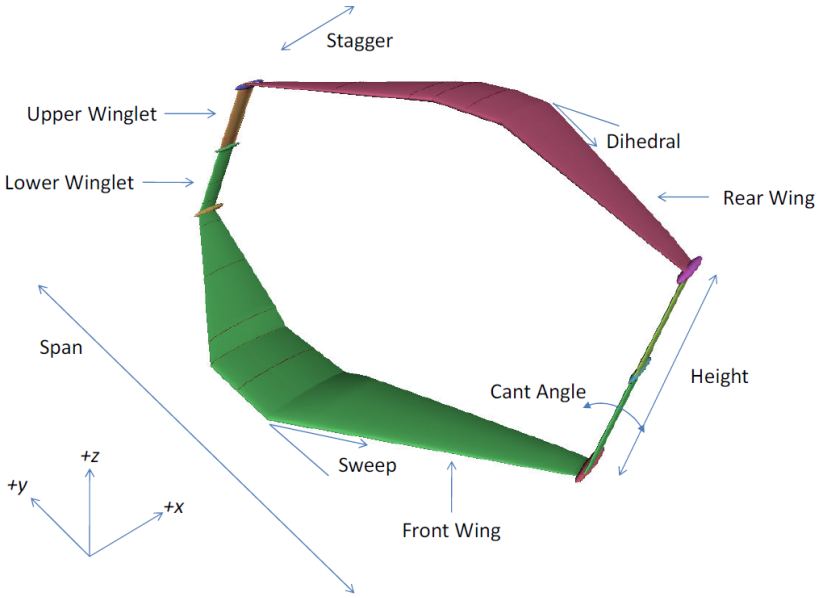


Figure 2.7: Box-wing geometry analyzed in Khan [26].

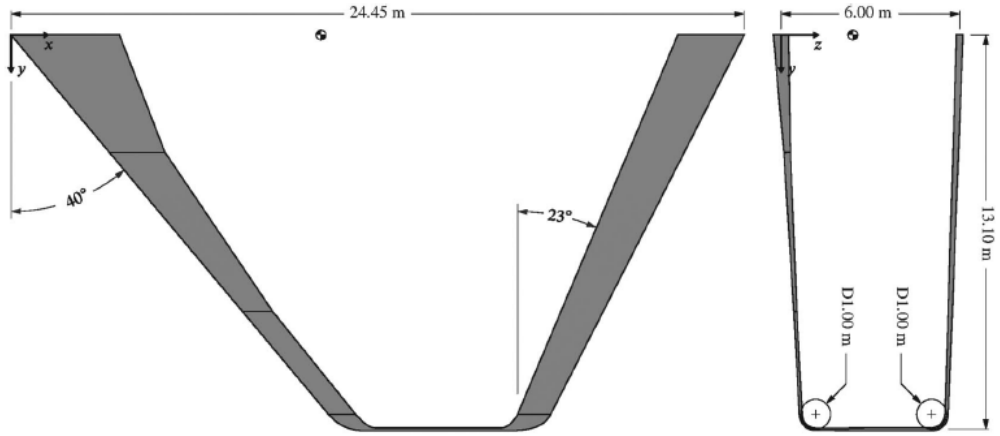


Figure 2.8: Box-wing geometry analyzed in Gagnon & Zingg [27].

the parameters tuned in this study, including tether length, rated airspeed, elevation angle, only the wings aspect ratio area and airfoils are directly involved with the kite geometry. Parameters such as the height to span ratio, the stagger or the cant angle are not part of this paper. However, the performance of the biplane resulted to be much higher in terms of lift coefficient than that of the monoplane. This work could be further improved by varying more geometrical parameters during the optimization, include CFD in the optimization or couple with a structural solver. Finally, although the proposed model (Figure 2.10) was not strictly a box-wing but a closed biplane with no lateral airfoils, it is of high relevance for this work that a similar concept has been studied for AWE systems with potential improvements on rigid kites.

2.6. SOCIECONOMIC IMPACT

In general, AWE systems present a rapid growth in the last decade and it is expected to continue growing in the following years. Several patents, prototypes and demonstrators have been built and research teams all over the world are working in several aspects including electronics, design and control of high-altitude wind energy kites [7]. It is relevant to remark that wind drones present several advantages when compared to wind turbines. The first one is the lower amount of resources, towers are replaced by thin tethers (6cm thick for 8 MW system) [6], wings need between 1%-10% the material required for a turbine blade [30] and time required for massive production is reduced. The second one is the feasibility of reaching higher altitude

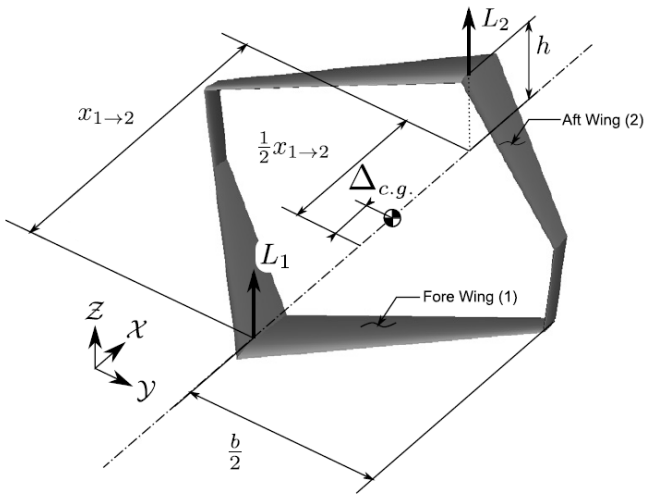


Figure 2.9: Box-wing geometry analyzed in Andrews & Perez [28].

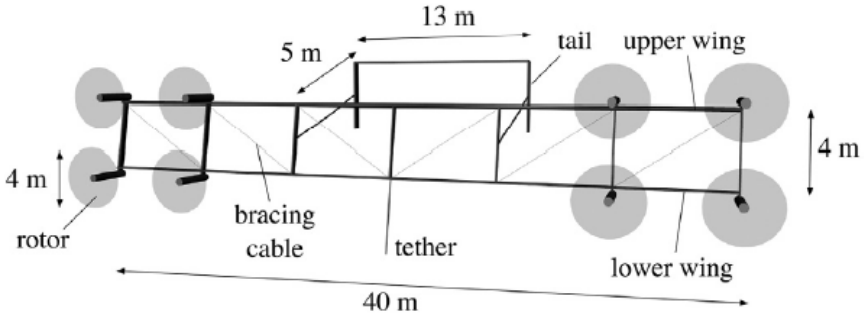


Figure 2.10: Biplane kite design proposed in Bauer et al. [29].

winds, and thus higher power density. The disadvantage of wind drones would come in terms of stability but this is compensated with autonomous systems that make them stay up in the air. The conclusion of this brief comparison is that wind drones can produce more energy while having cheaper costs than wind turbines.

In terms of social acceptance, several aspects are studied in [31] and are summarized hereafter. In particular, 5 different technological aspects are analyzed in detail: safety, visibility, sound emissions, ecological impacts and siting. Regarding safety, AWE systems are in general regarded as more hazardous than wind turbines. Some concerns include the movement of electric tethers through air in fly-gen AWE systems, increased risk to regular aviation traffic, or the unfeasibility of stopping the system if anything goes wrong. To mitigate these risks, a regulatory framework for safety must be present in the industry, the redundancy and tolerance to failure of the systems must be increased and the risk of accidents must be minimized below an acceptable threshold. Regarding visibility, wind kites impact is in general more positive than in the case of wind turbines. Taking into account that wind kites operate at higher altitudes, they produce lower or negligible shadow cast on the ground. Although the ground station might be a problem visually speaking, AWE systems lack of a tower structure needed for wind turbines, and thus reduce its visual footprint. Finally, the fact that they can be turned off when there is low or no wind also contributes to lower the visual concerns. Regarding acoustic emissions, in general it is stated that the impact is low because the wind drones operate at high altitudes. However, the real impact should be studied by including the tether and ground station noise and monitoring residents attitude close to AWE sites during long term operations. Regarding ecological aspects, the main concern is the collision of flying beings. In principle, the amount of bird strikes are expected to be lower than in the case of a wind turbine because of the higher operational height. It is also shown that birds are likely to collide with the tether without receiving any significant harm. However, there is still a lot of research to do on different ecosystems and seasons to conclude that the impact on flying animals is negligible or even

lower than in wind turbines. Regarding siting of AWEs, there are several drivers connected with the previously mentioned technological aspects. On the one hand, social acceptance is expected to be key in populated areas, this is why AWEs operating offshore or remote in areas would imply a higher social acceptance. On the other hand, other airspace operators are also reluctant to accept AWEs in an already busy airspace, not only in terms of capacity but also on safety. However, segregation of airspace coincide with the particular regulations of each country and a more detail study shall be made.

In terms of the economic growth, several achievements have been made in the last years. In 2020, China and the Netherlands had the highest capacity in terms of offshore wind installations, more than 3GW and 1.5 GW respectively. Being 70% of the global installed offshore capacity (34 GW) in Europe, member countries are leading both floating wind facilities and AWE demonstration projects [32].

According to the United Nations Climate Change Technology Executive Committee [33], companies developing and using AWEs have a level of maturity, known as Technological Readiness Level (TRL), between 3 and 8. Examples of these companies and different institutions are depicted in Figure 2.11. A relevant metric that companies consider before investing on AWE or any projects is the Levelized Cost of Energy (LCOE). It is defined as follows [11]:

$$LCOE = \frac{ICC \times CRF + OMC}{AEP} \tag{2.4}$$

Where ICC stands for Initial Capital Cost (EUR or USD), OMC for Operation and Maintenance Costs (EUR/y or USD/y), AEP for Annual Energy Production (MWh/y or kWh/y) and CRF for Capital Recovery factor depending on the discount rate i and the number of operational years n_y.

$$CRF = \frac{i(1+i)^{n_y}}{(1+i)^{n_y} - 1} \tag{2.5}$$

As long as the units are consistent, LCOE can be expressed in EUR/MWh or in USD/kWh. It may be around 0.23 USD/kWh for the first commercial systems and predicted to be 0.14 USD/kWh for 2030 which makes them cost competitive with diesel generated electricity systems whose LCOE is below 0.23 USD/kWh [34]. Potential markets as a function of AWEs size and autonomy, together with the deployment sequence are presented in Figure 2.12. Note that off grid application is suitable for countries or regions who lack of a good grid. However, in Europe, the most promising application would be offshore, not only because of the congested airspace and populated areas but the available space and the less restrictive safety requirements.

Finally, the main concerns on AWEs of different stakeholders are identified by means of interviews conducted by workers of the European Commission [11]. These stakeholders included academic, business and public personnel. Academic stakeholders are mainly concerned about the system readiness (31%), which includes continuous and autonomous operation of the systems and the economic performance (38%). Business stakeholders are concerned about the regulatory environment (25%) applicable to AWEs and the economic performance (24%). Public stakeholders express their concern towards funding availability (24%) to finance AWEs projects and the regulatory environment (29%) they need to develop.

2.7. REGULATIONS AND LEGAL FRAMEWORK

This work is not required to comply with any as it is focused on box wings aerodynamics in a simulation environment (CFD and panel methods). However, it was important to mention the regulations that would apply wind drone was physically manufactured for validation and testing in future projects.

Nowadays, there is no common framework that regulates AWES and prototypes are usually operating with special permits issued by the local aviation authorities. In Europe the European Aviation Safety Agency (EASA) is the authority responsible for the European aviation regulations. In general AWES are seen as a special case of Remotely Piloted Aircraft Systems (RPAS) and they must comply with safety regulations as RPAS do. It is important to remark that EASA becomes involved in the tests only if the vehicle is heavier than 150 kg. Otherwise it is the National Aviation Authority (NAA) the one responsible of regulating AWES [11] and issuing the Permits to Fly. The certification process which involves the agreement between the authority and the applicant, is done by means of a certification basis. This includes the Certification Specifications (CS), Special Conditions (SC) and the statement of acceptable Means of Compliance (MoC) using Aerospace Recommended Practices (ARP) standards [35].

According to US regulations [36] tethered aircraft are excluded from unmanned aircraft which means AWES are just regarded as air traffic obstacles and they follow a different regulation than for the European case.

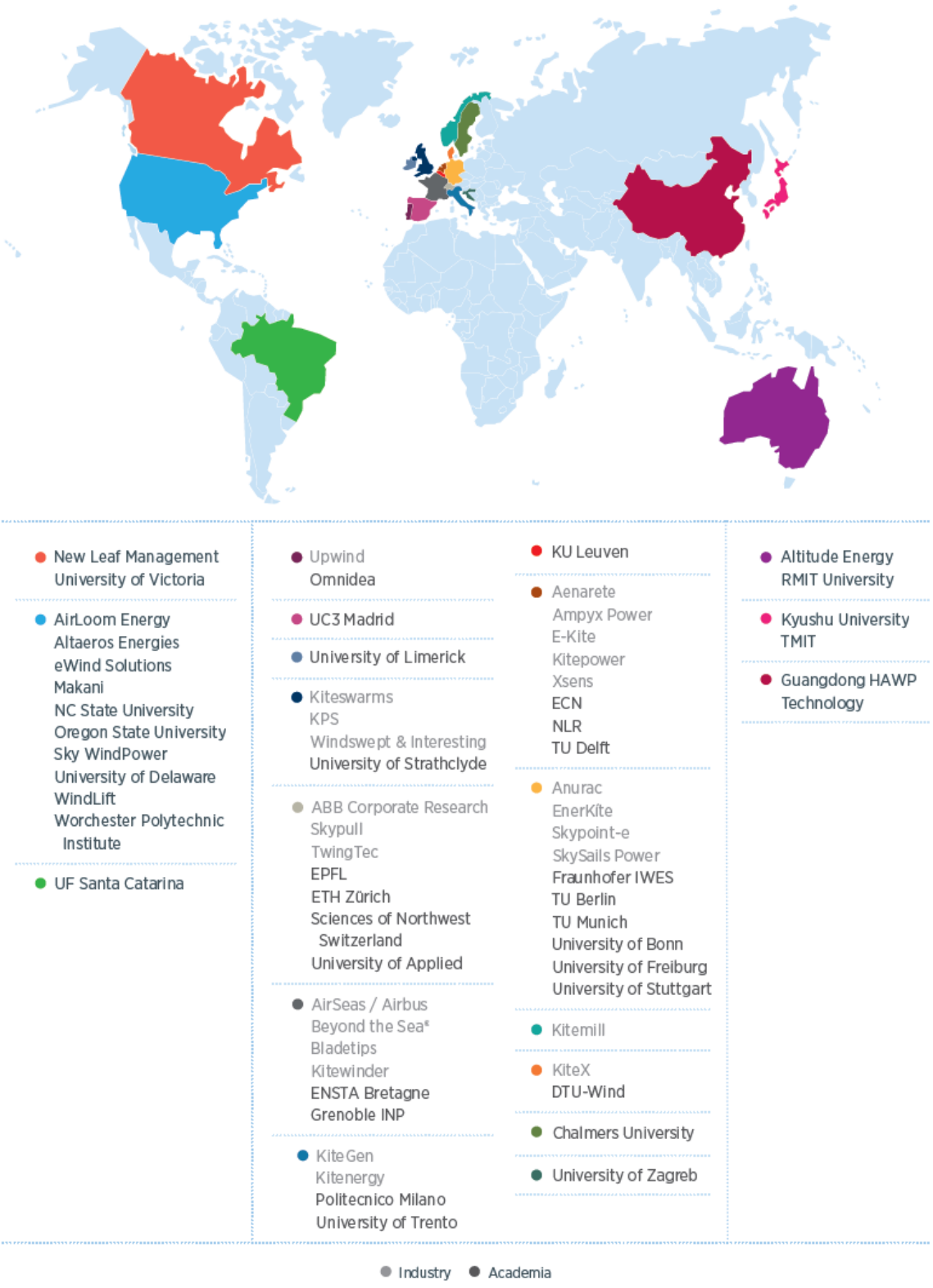


Figure 2.11: Airborne wind energy main contributors in 2021 [32].

In terms of the use of airspace, AWEs are expected to fly in Class G airspace which does not require any specific equipment nor control tower communication. Class G has a typical limit of 1200 feet (365 m) above

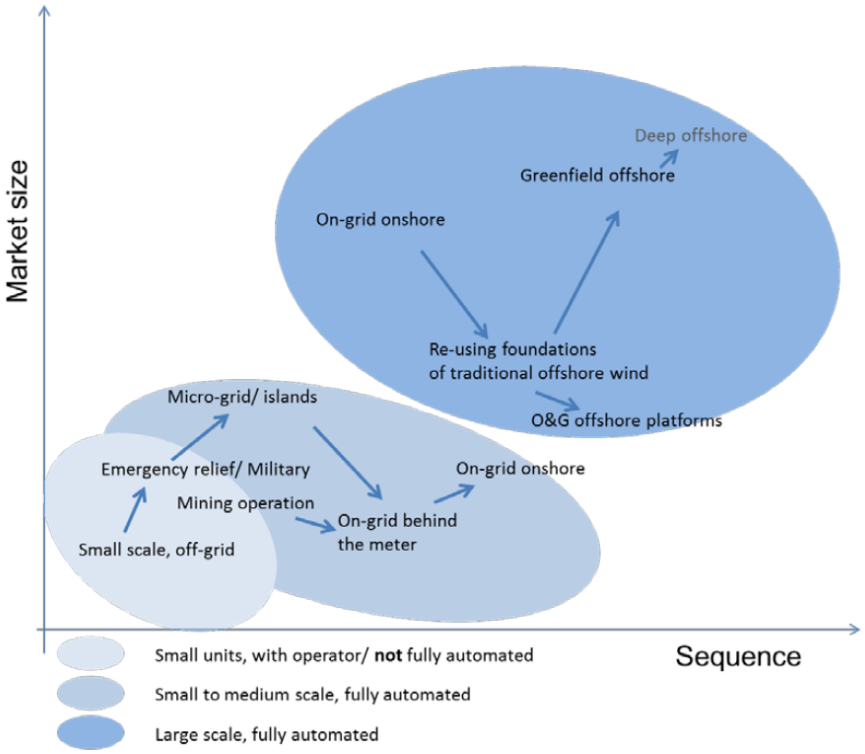


Figure 2.12: Potential markets for AWEs deployment [11].

ground level (AGL) that can be reduced down to 700 feet (213 m) AGL if there is an airport in the vicinity. However, even if it is an uncontrolled airspace, civil aviation rules must be followed and interference with other airspace users must be minimized [35].

3

OBJECTIVES AND METHODOLOGY

This chapter is focused on defining the key objectives of the present work, together with the tools needed to achieve them. Afterwards, the methodology used to reach these goals is explained with the aid of flowcharts, both the overall process and the particular process blocks.

3.1. OBJECTIVES

The ultimate goal of this work is the generation of reliable simulations to study the aerodynamics of box wing design concepts applied to AWE systems. In order to achieve this goal, several derived goals were identified:

1. Define the parameters relevant to box wing geometries.
2. Create a framework capable of performing fast aerodynamic computations (low reliability) of box wing designs for design and optimization purposes.
3. Create a framework to define the high quality CFD meshing process in an automatic manner.
4. Create auxiliary tools to minimize the user inputs and save time in pre and post processing of CFD (high reliability) simulations.
5. Integrate the previous frameworks and tools to maximize the performance of the computational resources and minimize the overall user and machine time in the loop.
6. Validate the developed tools with experimental data.
7. Define a box wing design equivalent to a conventional wing design for AWE systems applications together with the quantification of the deficiencies between both designs.
8. Visualize flow streamlines, compute the viscous drag and the different regions in the lift curve including stall.

3.2. TOOLS

In order to achieve the previously defined goals, several programmes were used. Note that these programmes are explained in detail in the following chapters of this work. However, find the summary of the programmes and their respective functions below:

1. APAME is a steady panel method software used for aerodynamic forces and moments calculations suitable for evaluation and optimization [37]. In this work it is associated with objectives 2 & 7.
2. Pointwise is a high quality meshing software developed for CFD preprocessing with a wide variety of applications, including aerospace [38]. In this work it is associated with objective 3.
3. OpenFOAM is an open-source software extensively used by academic organizations and industry for computational fluid dynamic simulations [39]. In this work it is associated to objectives 4 & 8.

4. MATLAB is a numerical computing environment suitable for data analysis, algorithms development data and modelling [40]. In this work it is the main integrator of the different programmes used and is associated directly or indirectly with all the objectives (1,2,3,4,5,6,7 & 8).

5. Paraview is an open-source software used for visualization and analysis applications [41]. In this work it is associated with objective 8.

3.3. METHODOLOGY

Having defined the objectives and tools used in this work, the general flow of the different processes involved is depicted in Figure 3.1. Shaded boxes mean they require some input or intervention from the user, whereas white boxes denote automated process with no user intervention. There are three main groups of processes within the flow chart separated by the two decision blocks.

1. Geometry generation and panel method solution

2. CFD mesh generation

3. CFD simulation and postprocessing

Notice that these three different groups are independent of each other and can operate separately provided the correct inputs for each of them.

Prior to generation of any innovative box wing design aerodynamic data, a validation of APAME and OpenFOAM applied to AWE box wings is required. This validation is indirectly validating the mesh resolution and quality used both in MATLAB and in Pointwise. The experimental data for the validation was found in Gall & Smith [21] and it is described in detail in the following chapters.

Overall, user tasks are divided in three and they include the declaration of parameters (geometrical, mesh, CFD, flight conditions), performance and quality checks and postprocessing. However, in this work there was an extra task which consisted of transferring the data and launching the simulations to the cluster. This would not be required provided that all programmes and resources were locally available.

3.3.1. GEOMETRY GENERATION AND PANEL METHOD SOLUTION

The first main process of this work entails a proper interaction between MATLAB and APAME. The final outputs of this process are the reference geometry and the aerodynamic performance. The reference aerodynamic design is the reference geometry fulfilling the criterion specified by the user in terms of aerodynamic performance.

Figure 3.2 depicts the detailed flow of processes involved in this first independent loop. The first step is to generate and store the box geometry in a MATLAB matrix having already defined the key parameters. This set of points define a wireframe geometry that is translated into nodes and geometrical panels connected accordingly to APAME format. Once the aerodynamic surface is meshed, the wake nodes and panels are generated and eventually connected to the trailing edge. All these commands together with the flight conditions are stored in an input file which is subsequently executed by APAME solver. Once the solver has finished, the C_L vs α curve and the reference geometry already stored in MATLAB are the outputs of this process.

This independent process can be used into three different ways depending on the application. For the validation of APAME tool, the reference aerodynamic design is fixed beforehand, so it is the user task to check the aerodynamic performance using the simulation results. For a parametric study, sweeping on different geometrical parameters and computing the aerodynamic quantities of interest is required. A final check from the user in terms of the sensitivity of this quantities to the variation of geometrical parameters is expected. In this application, there is no need for a reference geometry as output unless chosen by the user. For an optimization process, a cost function is defined together with some constraints and variables to be optimized. The loop is expected to run in this case until a certain minimum or maximum property is found. The reference aerodynamic design output of this application is the optimum design.

3.3.2. CFD MESHING

The second main process of this work entails a proper interaction between MATLAB and Pointwise. Figure 3.3 depicts the detailed flow of processes involved in this second independent loop. Pointwise is a bottom-up software so the meshing process runs incrementally. The first step consists on translating already existing wireframe data into points, curves and surfaces in Pointwise format. Next, the 1D mesh (nodes for Pointwise)

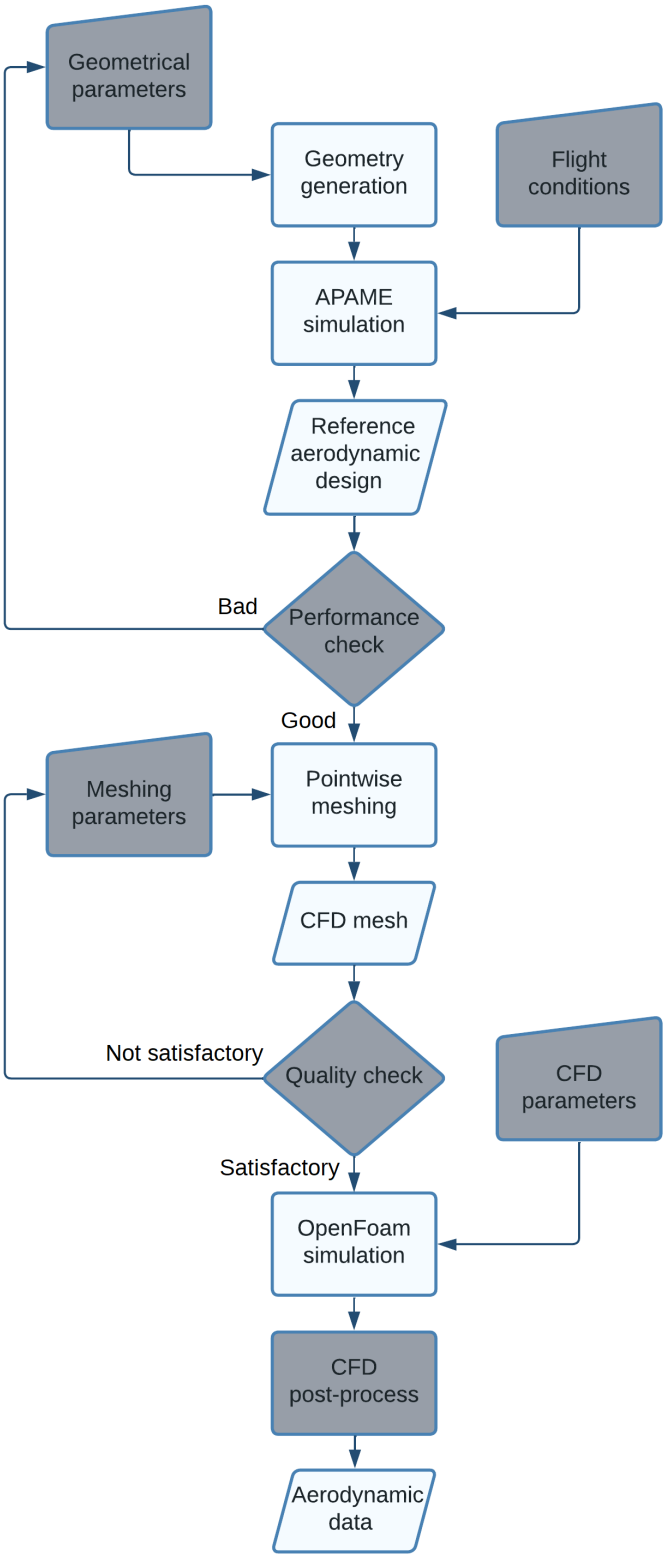


Figure 3.1: Flowchart of the whole process

is generated in the curves, and the 2D mesh (domains for Pointwise) is generated in the surfaces. Following this process, the farfield region is defined together with the refined regions for the 3D mesh. Then, the 3D mesh (block for Pointwise) is started having specified the meshing parameters already. All these actions are written by MATLAB in the form of commands into a native Pointwise scripting language called Glyph, an extension to Tcl [42]. Once it is ready, Pointwise is called and executes the glyph file with all the mesh specifications. It finishes by exporting directly the mesh in OpenFOAM format and in Pointwise format, so it can be checked afterwards. The quality criteria for the meshing process will be discussed in detail in further sections.

3.3.3. CFD SIMULATION AND POSTPROCESSING

The third main process of this work entails a proper interaction between MATLAB and OpenFOAM. Figure 3.4 depicts the detailed flow of processes involved in this third independent loop. With a CFD mesh ready, the only task of the user should be to define the CFD parameters. MATLAB will translate this user requirements into OpenFOAM configuration files. If both the mesh and these files are properly located, OpenFOAM solver will be initiated following a parallel decomposition of the mesh into the number of cores used for the computation. After the solver finishes, results coming from all the different processors are joined together. Note that for simulation in series, parallel decomposition and recomposition blocks do not exist. The outputs of these simulations are read by MATLAB for the case of the residuals and overall quantities and rendered with ParaView for the case of the physical magnitudes distributions.

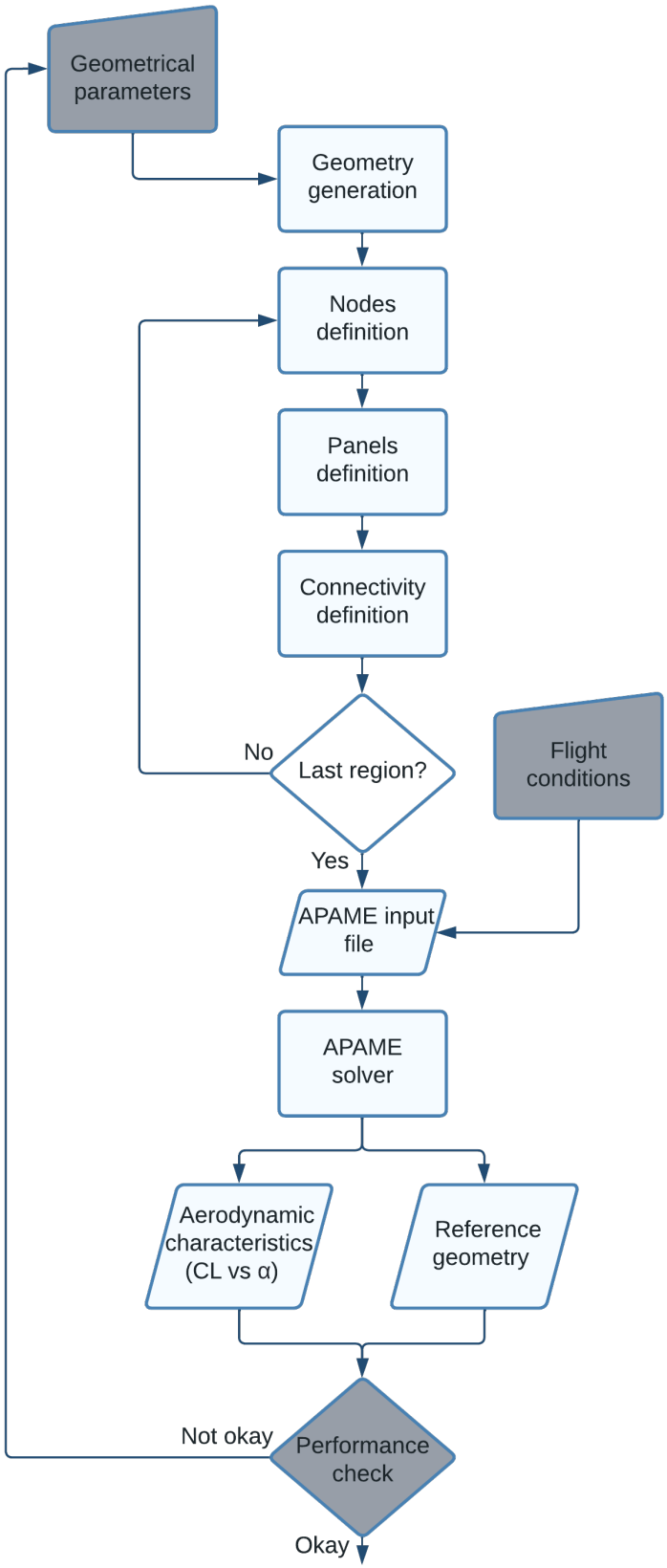


Figure 3.2: Flowchart of the geometry generation and panel method solution group

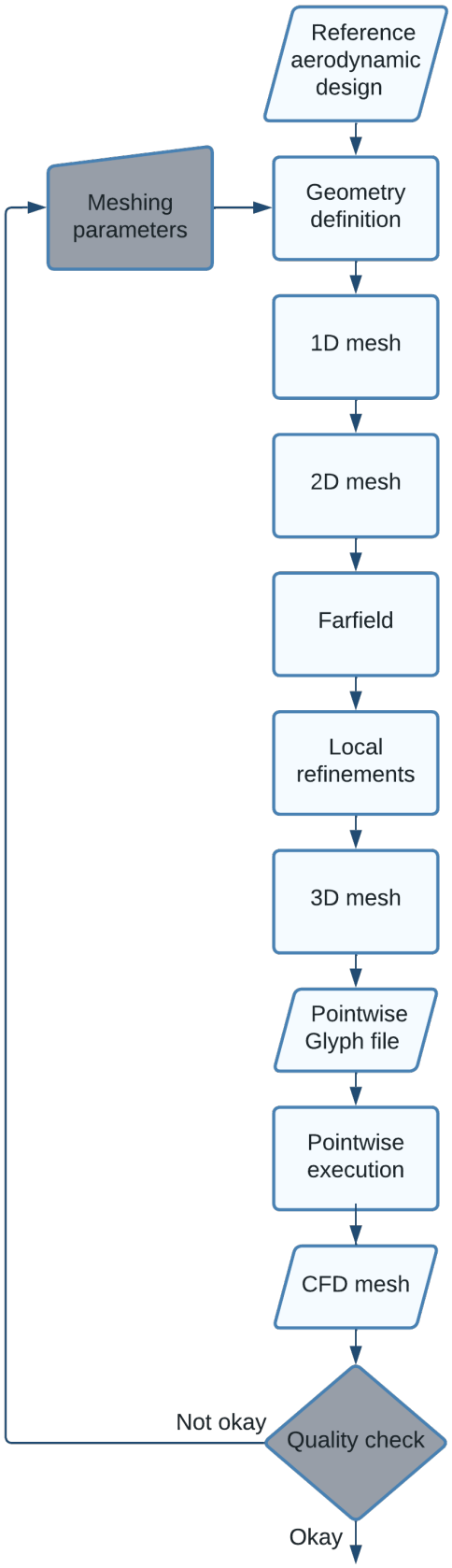


Figure 3.3: Flowchart of the CFD mesh generation group

3

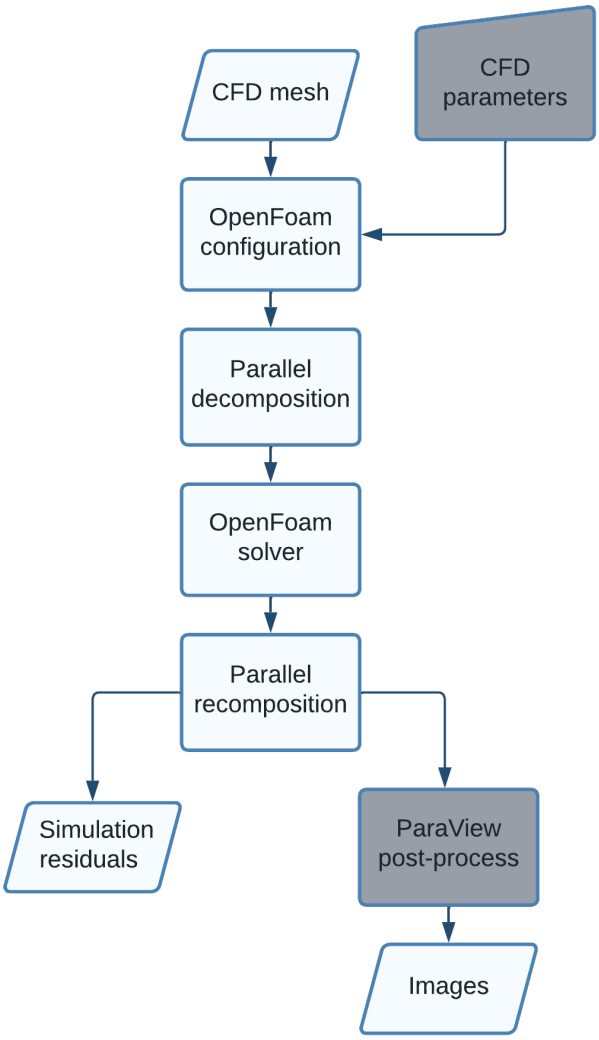


Figure 3.4: Flowchart of the CFD simulation and postprocessing group

4

COMPUTATIONAL FLUID DYNAMICS

This chapter focuses on describing the governing equations of fluid dynamics and discretizing them numerically for two different applications. The first one is focused on achieving a reference box-wing design optimizing with fast and low fidelity aerodynamic software. In this case, a brief introduction to panel methods is presented together with the software (APAME) that will be used in this work. The second one consists on having a high reliability estimation of the viscous drag present in box-wing rigid kites. A detailed description on Reynolds-averaged Navier-Stokes equations together with turbulence modelling approaches and the resolution algorithm implemented in OpenFOAM are presented in this chapter.

4.1. PANEL METHODS

As previously mentioned in the literature review, computational algorithms not including viscous drag estimation were used for box wing optimization purposes due to their fast performance and reasonable reliability on lift. The most popular method is called the panel method, which assumes the flow to be inviscid, incompressible, irrotational, and steady with the flow field described by the gradient of the velocity potential (Eq. 4.1).

$$\vec{u} = \nabla\phi \tag{4.1}$$

This potential has two fundamental boundary conditions: no wall penetration and Kutta condition at the trailing edge. Flat panels with an associated singularity are used to define the surface of the aerodynamic body. These singularities fulfill the Lapace equation on the velocity potential (Eq. 4.2) and can be a combination of sources, sinks, vortices and doublets. Wake panels need to be added at the trailing edge to satisfy the Kutta condition, which states flow must leave the trailing edge smoothly.

$$\nabla^2\phi = 0 \tag{4.2}$$

Examples of the computation of the velocity potential associated to single sources and doublets are given in Eqs. 4.3 & 4.4 respectively.

$$\phi(P) = \frac{\sigma_{SS}(Q)}{2\pi}\ln(r(P,Q)) \tag{4.3}$$

$$\phi(P) = \mu_D(Q)\frac{\partial}{\partial\vec{n}_Q}\left(\frac{1}{2\pi}\ln(r(P,Q))\right) \tag{4.4}$$

With the singularities positioned at some collocation point Q, and the velocity potential computed at a different point P, the strength of the source or sink σ_{SS}, the strength of the doublet μ_D and the direction of the doublet $\vec{n}_Q$. Once all panels and singularities are defined, it is possible to compute the strength of all singularities at each panel and thus compute the velocity potential. An example of the combined source and doublet panel method is formulated hereafter. The source strength at each panel is given by Eq. 4.5.

$$\sigma_j = \vec{n}_j \cdot \vec{Q}_\infty \tag{4.5}$$

With $\vec{n}_j$ the normal vector of that panel and $\vec{Q}_\infty$ the dynamic pressure. The boundary condition inside the surface at each collocation point is given by Eq. 4.6.

$$\sum_{j=1}^{N} B_j \sigma_j + \sum_{j=1}^{N} C_j \mu_j = 0 \tag{4.6}$$

With B_j, C_j the influence coefficients, and N the number of collocation points. Influence coefficients are usually computed assuming a unit strength singularity. However, the influence of the doublet panel on itself is computed using Eq. 4.7.

$$c_{ii} = \frac{1}{2} \tag{4.7}$$

Kutta condition provides Eq. 4.8.

$$(\mu_1 - \mu_N) + \mu_W = 0 \tag{4.8}$$

Combining this last equation with the influence matrix results in N+1 linear equations for the doublets influence.

$$\sum_{j=1}^{N+1} C_{ij}\mu_j = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1N} & c_{1W} \\ c_{21} & c_{22} & \dots & c_{2N} & c_{2W} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{N1} & c_{N2} & \dots & c_{NN} & c_{NW} \\ 1 & 0 & \dots & -1 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \\ \mu_W \end{pmatrix} \tag{4.9}$$

Operating on the previous matrix using Eq. 4.8 and relations on Eq. 4.10, it is possible to reduce it to order N.

$$\begin{aligned} a_{ij} &= c_{ij}, & j &\neq 1, N \\ a_{i1} &= c_{i1} - c_{iW}, & j &= 1 \\ a_{iN} &= c_{iN} + c_{iW}, & j &= N \end{aligned} \tag{4.10}$$

Eq. 4.6 finally yields the form:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_N \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ a_{21} & b_{22} & \dots & b_{2N} \\ \vdots & \vdots & & \vdots \\ b_{N1} & b_{N2} & \dots & b_{NN} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \end{pmatrix} = 0 \tag{4.11}$$

Since the source strength at any collocation point is known (Eq. 4.5), it is straightforward to compute the doublet strength with the previous relation. The velocity potential is then calculated using Eqs. 4.3 & 4.4. Having calculated the velocity potential, the velocity and the pressure are computed straightforward (Eqs. 4.1 & 4.12).

$$\frac{1}{2}\rho \left|\nabla\phi\right|^2 + p = constant \tag{4.12}$$

More details and a complete derivation of the panel method and its variants can be found in Katz & Plotkin [43].

For this particular work, the 3D panel method software that is used is called APAME [37], both because it is possible to run it without the graphical user interface, and the mesh can be scripted in a different software. It is an open-source program that is licensed under the General Public License. It is used for subsonic attached flows and suitable for optimization and conceptual design applications. It has been successfully used for AWE applications in different optimization frameworks and designs [44][45].

4.2. REYNOLDS-AVERAGED NAVIER-STOKES EQUATIONS

It is known that CFD methods rely on the Navier-Stokes equations, which are defined as follows.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \tag{4.13}$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \otimes \vec{u}) = \rho \vec{f}_e + \nabla \cdot \left(-p\bar{\bar{I}} + \bar{\bar{\tau}}\right) \tag{4.14}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot \left(\rho H \vec{u} - k_c \nabla T - \bar{\bar{\tau}} \cdot \vec{u} \right) = W_f + q_H \tag{4.15}$$

They represent mass, momentum and energy conservation respectively. Out of the three conservation laws, the energy equation Eq 4.15 is not fundamental for describing the velocity and pressure evolution. It is not used from this point onwards. In addition, flow is assumed to be incompressible, since the Mach number for AWE kites is usually below 0.3, thus density is constant. Viscosity in the freestream is also assumed to be constant, but not negligible. Under these simplifications, Eqs 4.13 & 4.14 simplify to:

$$\nabla \cdot \vec{u} = 0 \tag{4.16}$$

$$\rho \frac{\partial \vec{u}}{\partial t} + \rho \vec{u} \cdot \nabla \vec{u} = -\nabla p + \mu \Delta \vec{u} + \rho \vec{f}_e \tag{4.17}$$

Finally, external forces are neglected and steady flow is assumed for our particular aerodynamic studies, leading to Eq 4.17 being further simplified.

$$\vec{u} \cdot \nabla \vec{u} = -\frac{1}{\rho} \nabla p + \nu \Delta \vec{u} \tag{4.18}$$

Steady state Reynolds Averaged Navier-Stokes (RANS) equations are chosen for CFD simulations in this project. The reason is that even if the modelling complexity would be lowered, LES or DNS would require much more computational power. The starting point of the RANS equations is the Reynolds decomposition, which decomposes the velocity field into average and fluctuating part[1].

$$\vec{u}(\vec{x}, t) = \langle \vec{u}(\vec{x}, t) \rangle + \vec{u}'(\vec{x}, t) \tag{4.19}$$

A property regarding the fluctuation component states that $\langle \vec{u}'(\vec{x}, t) \rangle = 0$. Introducing Eq. 4.19 into Eqs. 4.16 & 4.18 and time averaging leads to RANS equations shown in Eqs. 4.20 & 4.21.

$$\nabla \langle \vec{u} \rangle = 0 \tag{4.20}$$

$$\nabla \cdot (\langle \vec{u} \rangle \otimes \langle \vec{u} \rangle) = -\frac{1}{\rho} \nabla \langle p \rangle + \nu \Delta \langle \vec{u} \rangle - \nabla \cdot \langle \vec{u}' \otimes \vec{u}' \rangle \tag{4.21}$$

Where $\langle \vec{u}' \otimes \vec{u}' \rangle$ is known as the Reynolds stress. It is a symmetric tensor containing 6 independent components and act as if there were additional stresses in the flow. The addition of this term leads to the well known closure problem of solving RANS. This problem appears from the fact that there are only 4 equations (1 for mass and 3 for velocity conservation) and 10 unknowns (pressure, 3 velocity components and 6 Reynolds stress tensor components). The way of approaching it is either by using turbulent viscosity models or Reynolds stress models. In this project turbulence viscosity models are used since they were found to be widespread used in the literature.

For the better comprehension of the reader, tensor notation is used from now onwards whenever possible, 4.20 & 4.21 are then expressed as Eqs. 4.22 & 4.23[2].

$$\langle u_i \rangle_{,i} = 0 \tag{4.22}$$

$$\left(\langle u_i \rangle \langle u_j \rangle \right)_{,j} = -\frac{1}{\rho} \langle p \rangle_{,i} + \nu \langle u_i \rangle_{,jj} - \langle u_i' u_j' \rangle_{,j} \tag{4.23}$$

4.3. DISCRETIZATION

In order to solve the Navier-Stokes equations numerically, it is necessary to transform them into discrete functions. This is achieved by dividing the space into nodes or cells and solving the discretized equations on these locations. There are usually three basic discretization techniques regarding the spacial derivatives: the finite difference method (FDM), the finite element method (FEM) and the finite volume method (FVM) [46].

The technique that is used in this work is the finite volume method. It consists on dividing the space into small volumes associating this volumes to each mesh point. This method is based on cell-averaged values and since integral formulation is used it has the advantage of quantities being conserved locally and globally.

[1] Mean quantities are denoted between $\langle q \rangle$, whereas fluctuations are denoted with q'

[2] Note that $_{,j}$ and $_{,jj}$ mean the first and second spatial derivative with respect to the coordinate x_j

It is also chosen because it can handle any type of mesh either structured or unstructured giving a higher amount of flexibility to the meshing process [47].

Each of the volumes resulting from the discretization process is treated as a control volume itself. As stated before, the mean values are computed for each control volume, but the flux calculation on each volume face has to be independent of the volume to ensure a conservative discretization.The evaluation of the cell face flux can be done using different schemes, but any of them will introduce errors in the solution that will increase the diffusion of the scheme. The particular FVM schemes for this work are described in chapter 5.

4.4. TURBULENCE MODELLING

In order to solve for the closure problem described in the previous section, models based on turbulent viscosity (Boussinesq) hypothesis are used [48]. Boussinesq hypothesis states that the turbulent stresses must be related to the mean velocity gradients in a similar way viscous stresses are related to the full velocity gradients. This hypothesis is shown in Eq. 4.24, where ν_t is the Boussinesq Eddy viscosity (or turbulent viscosity), δ_{ij} is the Kronecker delta and k is the turbulent kinetic energy.

$$\langle u_i' u_j' \rangle = -\nu_t \left(\langle u_i \rangle_{,j} + \langle u_j \rangle_{,i} \right) + \frac{2}{3} \delta_{ij} k \tag{4.24}$$

The first method to model the turbulent viscosity is called the mixing-length model and was proposed by Prandtl [49]. It is based on the concept of turbulent length scale l_m which is dependent on each particular problem and serves as a rough approximation to compute the Eddy viscosity through the algebraic expression described in Eq. 4.25.

$$\nu_t = l_m^2 \left| \langle u_i \rangle_{,j} \right| \tag{4.25}$$

This model has the advantage of being numerically efficient, as algebraic expressions can be easily computed by any machine. The drawbacks are that the turbulent velocity scale is determined completely by mean flow and that the mixing length is flow dependent.

An improvement was made to this zero equation model (algebraic) making it a one equation model known as the Turbulent Kinetic Energy model (TKE) [50] & [51]. Turbulent viscosity (Eq. 4.26) becomes now dependent on the turbulent kinetic energy k, which is computed from the transport equation (Eq. 4.27).

$$\nu_t = 0.55 l_m \sqrt{k} \tag{4.26}$$

$$\frac{\bar{D}k}{\bar{D}t} + \langle u_i' u_j' \rangle \langle u_i \rangle_{,j} = -\left(\frac{1}{2} \langle u_i' u_i' u_j' \rangle + \langle u_j' p' \rangle / \rho - \nu k_{,j} \right)_{,j} - \tilde{\epsilon} \tag{4.27}$$

TKE equation (Eq. 4.27) can be rewritten in a more compact way as in Eq. 4.28, where P denotes the production term and $\tilde{T}'$ is the model for flux term from gradient-transport hypothesis.

$$\frac{\bar{D}k}{\bar{D}t} - P = -\tilde{T}' - \tilde{\epsilon} \tag{4.28}$$

Note that the production term is closed though Boussinesq hypothesis, there is an algebraic dissipation model for $\tilde{\epsilon}$ (Eq. 4.29) and a model for the energy flux $\tilde{T}'$ (Eq. 4.30).

$$\tilde{\epsilon} = C_D k^{3/2} / l_m \tag{4.29}$$

$$\tilde{T}' = -\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \tag{4.30}$$

Further details of this model are out of the scope of this project. However, it is concluded that the model is not globally applicable since the length scale is still dependent on the flow.

An alternative for this model are models with a transport equation for turbulent viscosity. The most famous one is Spallart & Allmaras developed in 1992 [52]. It is a one equation model which models the Eddy turbulent viscosity in an analogous way to the convection-diffusion equation plus a source term. Eq. 4.31 includes, in the source term, the mean flow rotation $\bar{\Omega}$, the wall distance l_w, which is relevant for the behaviour near the wall, and a destruction term $|\nabla \nu_t|$.

$$\frac{\bar{D}\nu_t}{\bar{D}t} = \nabla \cdot \left(\frac{\nu_t}{\sigma_\nu} \nabla \nu_t \right) + S_\nu \left(\nu, \nu_t, \bar{\Omega}, l_w, |\nabla \nu_t| \right) \tag{4.31}$$

It is a model that was specifically calibrated for aerodynamic applications and it gives reasonable predictions for attached boundary layers. However, there are discrepancies found in separated flow and it is not satisfactory in some free shear flows either.

In order to make turbulent models universally applicable, two-equation models were developed. They are models using the Boussinesq hypothesis (Eq. 4.24) together with transport equations for 2 different turbulent scales (ϕ,ψ). The turbulent viscosity is calculated using these scales $\nu_t \sim \phi^n \cdot \psi^m$ and the power coefficients are selected via dimensional analysis.

The $k-\epsilon$ turbulence model is the most widespread model used extensively in commercial applications [53]. It was created by Jone & Launder in 1972 [54] but many modifications have been made regarding the parameters and coefficients present in the equations. The variation implemented in OpenFOAM is called the Standard $k-\epsilon$ model proposed by Launder and Spalding in 1974 [55].

The two scales in this model are the turbulent kinetic energy k and the turbulent dissipation rate ϵ which are modelled using the transport Eqs. 4.32 & 4.33. The expression for the turbulent viscosity is found in Eq. 4.34.

$$\frac{\bar{D}k}{\bar{D}t} = \left(\left(\nu + \frac{\nu_t}{\sigma_k}\right)k_{,j}\right)_{,j} + \tau_{ij}\langle u_i\rangle_{,j} - \epsilon \tag{4.32}$$

$$\frac{\bar{D}\epsilon}{\bar{D}t} = \left(\left(\nu + \frac{\nu_t}{\sigma_\epsilon}\right)\epsilon_{,j}\right)_{,j} + \tau_{ij}\langle u_i\rangle_{,j}\frac{C_{\epsilon_1}\epsilon}{k} - \epsilon\frac{C_{\epsilon_2}\epsilon}{k} \tag{4.33}$$

$$\nu_t = \frac{C_\mu k^2}{\epsilon} \tag{4.34}$$

Where the Reynolds stress tensor is formulated in Eq. 4.35, with S_{ij} the mean shear rate tensor. All the coefficients are described in Table 4.1. Note that initial and boundary conditions need to be specified for each particular problem. In addition, for steady simulations the temporal terms are dropped.

$$\tau_{ij} = 2\nu_t\left[S_{ij} - \frac{1}{3}\langle u_k\rangle_{,k}\delta_{ij}\right] - \frac{2}{3}k\delta_{ij} \tag{4.35}$$

$\mathbf{C}_\mu$	$\sigma_\mathbf{k}$	σ_ϵ	$\mathbf{C}_{\epsilon_1}$	$\mathbf{C}_{\epsilon_2}$
0.09	1	1.3	1.44	1.92

Table 4.1: Standard $k-\epsilon$ coefficients

This turbulence model is suited for external aerodynamics where no separation nor strong pressure gradients are expected. However the standard version is not directly applicable to wall flows and thus not accurate enough for predicting the flow turbulence in the boundary layer [56].

An alternative to this model is the popular $k-\omega$, which yields to more accurate results in wall bounded flows with adverse pressure gradients [53]. The two scales in this model are the turbulent kinetic energy k and the specific dissipation ω which are modelled using the transport Eqs. 4.36 & 4.37. The expression for the turbulent viscosity is found in Eq. 4.38 and the Reynolds stress tensor was defined in Eq. 4.35.

$$\frac{\bar{D}k}{\bar{D}t} = \left(\left(\nu + \sigma^*\nu_t\right)k_{,j}\right)_{,j} + \tau_{ij}\langle u_i\rangle_{,j} - \beta^* k\omega \tag{4.36}$$

$$\frac{\bar{D}\omega}{\bar{D}t} = \left(\left(\nu + \sigma\nu_t\right)\omega_{,j}\right)_{,j} + \tau_{ij}\langle u_i\rangle_{,j}\frac{\gamma\omega}{k} - \beta'\omega^2 \tag{4.37}$$

$$\nu_t = \gamma^*\frac{k}{\omega} \tag{4.38}$$

All the required turbulent model constants are depicted in Table 4.2.

β^*	σ^*	β'	σ	γ
0.09	0.5	3/40	0.5	5/9

Table 4.2: $k-\omega$ coefficients

Although the performance of this model in boundary layer flows is increased, it is quite sensitive to the freestream conditions set for k and ω, leading to changes in the solution and requiring validation with experimental data for each particular case [53].

A comparison of the previous three turbulence models was found in Wilcox [56] for different experiments in terms of spreading rate. Results are depicted in Table 4.3. Regarding the Spallart-Allmaras model, the plane jet and the rounded jet values lie outside of the corresponding experiments range. The reason is that this model was created for external aerodynamic applications like the flow past a wing, but not for applications including free shear jets expelled from a nozzle. Out of the two-equation models, $k-\omega$ is the closest to the range of experimental spreading rates and all the values lie within the experimental ranges but the mixing layer case. The $k-\epsilon$ model only predicts the jet plane case within the range of measured values. These results are in accordance with the goal of these models, which is provide a greater universality in modelling turbulent flows. In fact, by using the average of the measured values, the error between the models and experiments is 6% and 17% for $k-\omega$ and $k-\epsilon$ respectively.

	k − ϵ model	**k − ω model**	**SA model**	**Measured**
Plane wake	0.256	0.326	0.341	0.32-0.40
Mixing layer	0.098	0.096	0.109	0.103-0.120
Plane jet	0.109	0.108	0.157	0.10-0.11
Round jet	0.12	0.094	0.248	0.086-0.096

Table 4.3: Computed spreading rate of free shear flows

In order to alleviate the deficiencies of $k-\epsilon$ and $k-\omega$ turbulence models, the $k-\omega$ SST model was born in 1993 by Menter [57]. The basic principle of this model is to use the $k-\omega$ model near the wall, whereas the $k-\epsilon$ model is used far from the wall. The transition between the two models is determined by a blending function. The model implemented in OpenFOAM is the enhanced version published in 2003 and presented in [58]. The main equations describing this model are formulated in Eqs. 4.39 & 4.40.

$$\frac{\tilde{D}k}{\tilde{D}t} = \tilde{P}_k + \left(\left(\nu + \nu_t\sigma_k\right)k_{,j}\right)_{,j} - \beta^* k\omega \tag{4.39}$$

$$\frac{\tilde{D}\omega}{\tilde{D}t} = \gamma\frac{\tilde{P}_k}{\nu_t} + \left(\left(\nu + \nu_t\sigma_\omega\right)\omega_{,j}\right)_{,j} - \beta'\omega^2 + 2\left(1-F_1\right)\sigma_{\omega 2}\frac{1}{\omega}k_{,j}\omega_{,j} \tag{4.40}$$

Note that these equations contain a limited production term $\tilde{P}_k$ for both the kinetic energy and the specific turbulent dissipation rate. This term prevents turbulence modelling in stagnation regions. This is given by Eq. 4.41, where P_k is formulated in 4.42.

$$\tilde{P}_k = \min\left(P_k, 10\beta^* k\omega\right) \tag{4.41}$$

$$P_k = \nu_t\left\langle u_i\right\rangle_{,j}\left(\left\langle u_i\right\rangle_{,j} + \left\langle u_j\right\rangle_{,i}\right) \tag{4.42}$$

They also contain two different blending functions: one regarding the transition between models (F_1) and another one regarding the Eddy viscosity (F_2). The first blending function is formulated in Eq. 4.43, with CD_{kw} specified in Eq. 4.44 and y the nearest distance to the wall.

$$F_1 = \tanh\left\{\left\{\min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right), \frac{4\rho\sigma_{w_2}k}{CD_{kw}y^2}\right]\right\}^4\right\} \tag{4.43}$$

$$CD_{kw} = \max\left(\left(2\rho\sigma_{w_2}\frac{1}{\omega}k_{,j}\omega_{,j}\right), 10^{-10}\right) \tag{4.44}$$

Note that F_1 is zero away from the wall ($k-\epsilon$) and switches to one inside the boundary layer ($k-\omega$). This has a direct influence in model constants, which are interpolated between the two models following Eq. 4.45.

$$\alpha = \alpha_1 F_1 + \alpha_2(1-F_1) \tag{4.45}$$

The second blending function appears in the definition of the turbulent Eddy viscosity (Eq. 4.46).

$$\nu_t = \frac{a_1 k}{\max\left(a_1\omega, |S|F_2\right)} \tag{4.46}$$

Here S denotes an invariant measure of the strain rate. The second blending function F_2 is formulated in Eq. 4.47.

$$F_2 = \tanh\left\{\left[\max\left(\frac{2\sqrt{k}}{\beta^*\omega y}, \frac{500\nu}{y^2\omega}\right)\right]^2\right\} \tag{4.47}$$

All the required turbulent model constants are depicted in Table 4.4.

β_1	γ_1	σ_{k1}	σ_{k2}	β_2	γ_2	σ_{ω_1}	σ_{ω_2}	β^*	a_1
0.075	5.0	0.85	1.0	0.0828	0.44	0.5	0.856	0.09	0.31

Table 4.4: $k-\omega SST$ coefficients

4.5. SOLVING THE EQUATIONS IN OPENFOAM

Since the main purpose of this work is to compare the aerodynamic characteristics of box wing and conventional wing designs by using the C_L vs α and the drag polar curves, a steady state solver is used. In particular, OpenFOAM has a steady state incompressible solver that uses the Semi-Implicit Method for Pressure Linked Equations, known as SIMPLE [59]. The basic steps of this method are described hereafter [60][61]:

1. Set the boundary conditions for pressure (p^{old}) and velocity fields ($\vec{u}^{old}$) or use the values from previous iteration.
2. The velocity field is approximated by solving the momentum equation (Eq. 4.18). Note that this approximation ($\vec{u}^{p}$) does not necessarily fulfill the continuity equation (Eq 4.16). The pressure gradient term is computed by using the pressure field from step 1. The equation is implicitly under relaxed (see Eq. 4.48) by using the velocity under-relaxation factor α_v.

$$\vec{u}^{new} = \vec{u}^{old} + \alpha_v\left(\vec{u}^{p} - \vec{u}^{old}\right) \tag{4.48}$$

3. Compute the mass fluxes at the cell faces.
4. The Poisson equation for pressure is solved to obtain the new pressure distribution. Note that this equation results from operating continuity and momentum equations (Eqs. 4.16 & 4.18).

$$\frac{1}{\rho}\nabla^2 p^p = -\nabla\cdot\left(\vec{u}^{new}\cdot\nabla\vec{u}^{new}\right) \tag{4.49}$$

   Note that under relaxation is applied on this step to take into account convection-diffusion error. The pressure under-relaxation factor α_p is used in this case, together with the solution of the pressure equation p^p.

$$p^{new} = p^{old} + \alpha_p\left(p^{p} - p^{old}\right) \tag{4.50}$$

5. Correct the mass fluxes at the cell faces.
6. Correct the velocity field using the pressure field from step 4, aiming to satisfy the continuity equation (Eq. 4.16).
7. Update the boundary conditions on the pressure and velocity fields.
8. Repeat the whole process until convergence, which means velocity field satisfying both the continuity and momentum equations (Eqs. 4.16 & 4.18).

Non-orthogonality convergence issues can be reduced by repeating steps 4 and 5 for a prescribed number of times. These are called non orthogonal correctors. All turbulent scaling transport equations (k,ω) are solved within the loop after the corrector step (step 6). There is a variant of this algorithm called the SIMPLE consistent (SIMPLEC) algorithm that enhances faster convergence with minimal modifications. In particular, two modifications are made to the SIMPLE algorithm.

1. When discretizing the momentum equation in a control volume (cell), there are some terms called the neighbouring terms that are neglected by the SIMPLE algorithm. SIMPLEC algorithm keeps these terms but computes them by means of approximations instead of using the exact formulas.

2. Pressure relaxation factor is set to $\alpha_p = 1$, which means no relaxation in the pressure field is required.

For a detailed explanation of this variant, please refer to Vaan Doormaal & Raithby [62].

5

VEHICLE DEFINITION, MESH AND INITIAL SETUP

This chapter aims to give a detailed description of the preprocessing of the geometry. In particular, the geometry for software validation is used. The chapter begins with the definition of the most relevant variables for box wing design. It continues with the description of the meshing process and validation in APAME. Afterwards, there is a detailed description of the meshing process and quality metrics used in Pointwise, together with the different meshes for the resolution study. Finally, OpenFOAM settings are explained, ending with the comparison to experiments of OpenFOAM RANS simulations.

5.1. BOX-WING PARAMETERS

Having reviewed the parameters that are used for box-wing design in Chapter 2, the following parameters are considered relevant for the present work and are depicted in Figure 5.1:

- The wingspan denoted as b.
- The vertical separation between wings called height and denoted as h.
- The horizontal separation between wings called stagger and denoted as s.
- The chords at four different sections of the wing: lower wing root and tip (c_{r1} and c_{t1}), and upper wing root and tip (c_{r1} and c_{t1}). Note that the chord at the winglet is determined by the chord at the respective wing tips (c_{t1} and c_{t2}) which is kept constant during the rotation and interpolated in the straight region.
- The capability of having five different airfoils at the previously mentioned sections: upper wing root, upper wing tip, lower wing root, lower wing tip, and winglet.
- The radius defining the joints between wings and winglet called cant radius and denoted as R. This radius refers to the rotation of the whole sections planes. In order to joint the points between sections, it is necessary to compute local centers of rotation. The center of rotation is strongly dependent on the points that are being rotated from wing to winglet or viceversa. For example the center of rotation of the leading edge between the lower wing tip section and winglet section lies on the line defined by the unitary vector $\vec{h}$ (not to scale) at a distance R from the lower wing tip LE. For more details about the computation of the center of rotation refer to Annex A.1. In this work R is parametrized as a percentage of the winglet length ($\sqrt{h^2+s^2}$), so that it is independent on the airfoil profiles at the tip.
- Five different parameters specifying spanwise and chordwise divisions of the main surfaces, which are defined for half the box wing. N_{bl}, N_{bu} and N_{bw} are the number of spanwise sections for the lower wing, upper wing and winglet respectively. Note that physically, they could be interpreted as the position where the ribs would be located. N_{af} is the number of chordwise points to discretize the airfoil geometry. Finally N_R is the number of points to describe the curved regions.

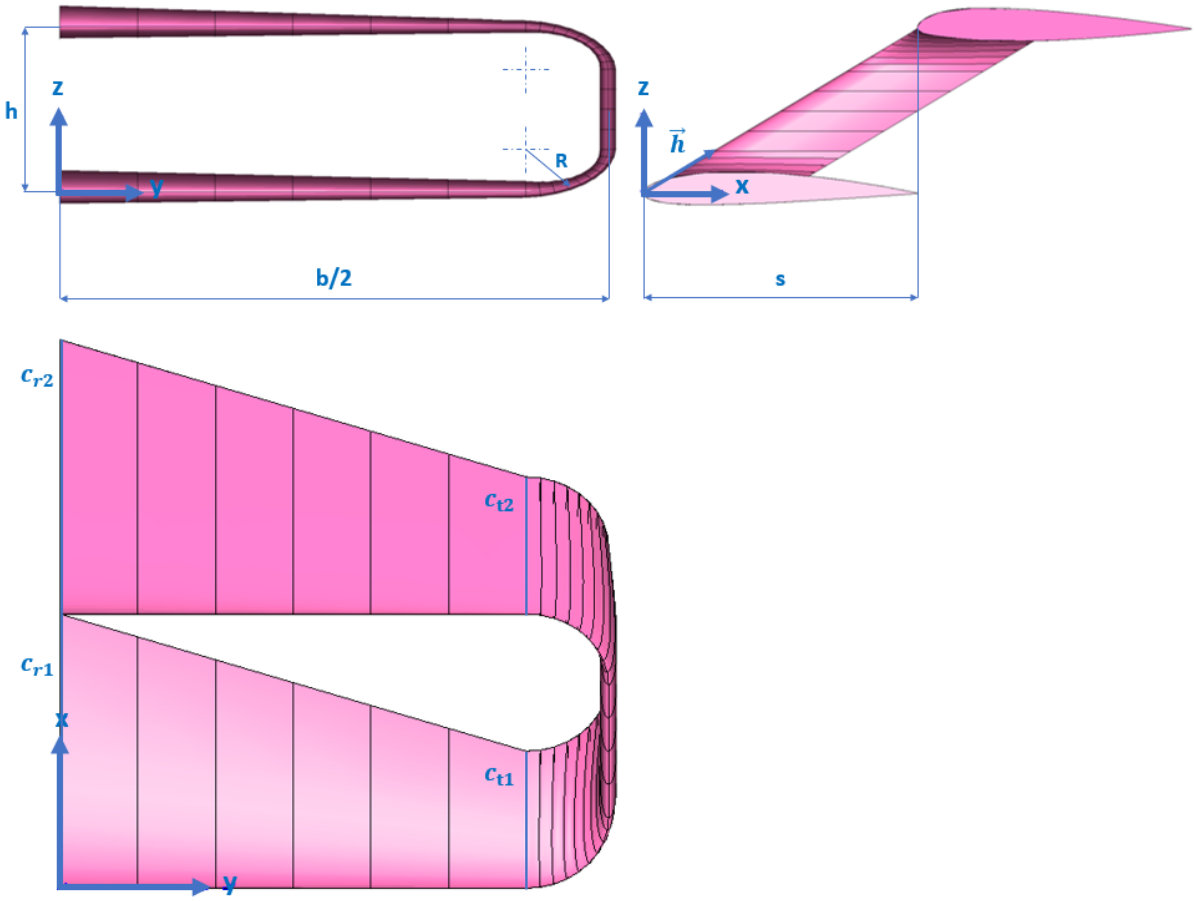


Figure 5.1: Geometry definition and parameters

All the geometry points are generated and stored in a MATLAB matrix. This data is used for two different meshing processes. The first one is generating the mesh for APAME directly from MATLAB. The second one is to generate the necessary Pointwise scripting file containing the commands to generate the CAD model and subsequent CFD meshing.

5.2. VALIDATION SET

As mentioned in chapter 3 both APAME simulations, Pointwise meshes and OpenFOAM simulations need to be validated with experimental data. This data, applied to a particular box wing design, was found in Gall & Smith [21].

Table 5.1 contains all the geometrical parameters of the box wing under examination, whereas Table 5.2 specifies the flight conditions. This particular box wing has constant chord and airfoil in all sections except for the winglet, being this value equal to the mean aerodynamic chord (MAC). The value for the cant radius was not found in the paper but it was assumed to be 15% of the winglet length. In addition, the stagger of this wing is not specified to be forward or backwards. Due to time constraints, to avoid performing two mesh resolution studies (for CFD), this stagger was assumed to be 0. Figure 5.2 depicts half of the wing generated and stored in MATLAB and Table 5.3 contains the measured aerodynamic data with its corresponding uncertainty.

5.3. APAME MESH

Although APAME could handle unstructured meshes, a structured mesh consisting of panels aligned with the flow velocity was chosen for better performance. Figure 5.3 shows an example of the full box wing meshed for APAME. However, the panel resolution needs to be set to a fixed value. This value should provide reliable

b [**m**]	1.0160
c [**m**]	0.2032
h [**m**]	0.2032
s [**m**]	0
Airfoil	NACA 0012
Airfoil winglet	NACA 0003
R [%]	15

Table 5.1: Geometrical parameters

Re	510000
$\mathbf{T}_\infty$ [**K**]	288.15
ρ_∞ [$\mathbf{kg/m^3}$]	1.228
ν [**kg/ms**]	1.4603E-5

Table 5.2: Reference conditions

α [°]	$\Delta\alpha$ [°]	C_L	ΔC_L	C_D	ΔC_D
-2	± 1	-0.0851	± 0.02	0.0182	± 0.0005
0	± 1	0.0125	± 0.02	0.0173	± 0.0005
2	± 1	0.0700	± 0.02	0.0182	± 0.0005
4	± 1	0.2165	± 0.02	0.0239	± 0.0005
6	± 1	0.3541	± 0.02	0.0358	± 0.0005
8	± 1	0.5273	± 0.02	0.0611	± 0.0005
10	± 1	0.6605	± 0.02	0.0892	± 0.0005
12.3	± 1	0.7625	± 0.02	0.1151	± 0.0005
14.5	± 1	0.7933	± 0.02	0.1503	± 0.0005
16.5	± 1	0.8374	± 0.02	-	-
18.5	± 1	0.8549	± 0.02	-	-
21	± 1	0.7787	± 0.02	-	-

Table 5.3: Experimental data

results in the least amount of time possible.

5.3.1. APAME MESH RESOLUTION STUDY

A mesh resolution study was conducted with five different refinement levels: 1972, 3828, 7540, 15196 and 30276 panels, built in as indicated in Table 5.4. Eq. 5.1 allows to compute the number of panels given the spanwise and chordwise divisions.

$$Panels = 2\left[(N_{bl}+1)+(N_{bu}+1)+(N_{bw}+1)+2\left(N_R-1\right)\right]\left(N_{af}-1\right) \tag{5.1}$$

Knowing the limits of panel methods two decisions were taken. In the first place, only the lift coefficient data is compared with experimental data. In the second place, only data in the linear region is checked during the comparison. Figure 5.4a shows the chosen points for this comparison and the qualitative trend. However, the relative error was computed for a quantitative comparison in the following way:

$$Relative\ error\ (\%) = 100 \cdot \left| \frac{C_{L,APAME} - C_{L,Experimental}}{\max\left(C_{L,Experimental}\right)} \right| \tag{5.2}$$

Having computed the relative error, some conclusions can be extracted. The first of them is that regardless of the number of spanwise panels, the value of the lift coefficient experiences almost no change for a given angle of attack. The second one is that excluding $\alpha = 2°$, the lift coefficient that APAME provides is clower than the experimental value. The final one is that the lowest panel resolution is the one providing the least relative error on average. Note that choosing this panel resolution is advantageous since the computational time increases roughly proportional to the number of panels squared. However, the fact of the stagger being

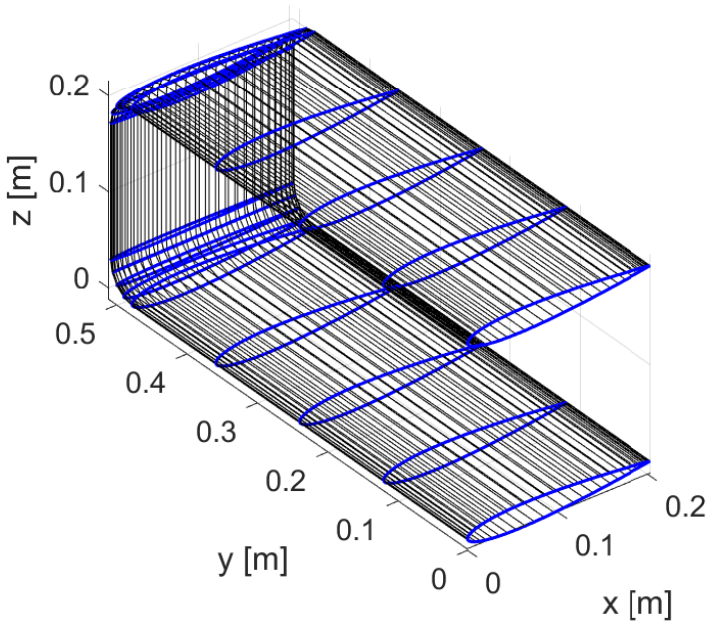


Figure 5.2: Box wing parametrization in MATLAB

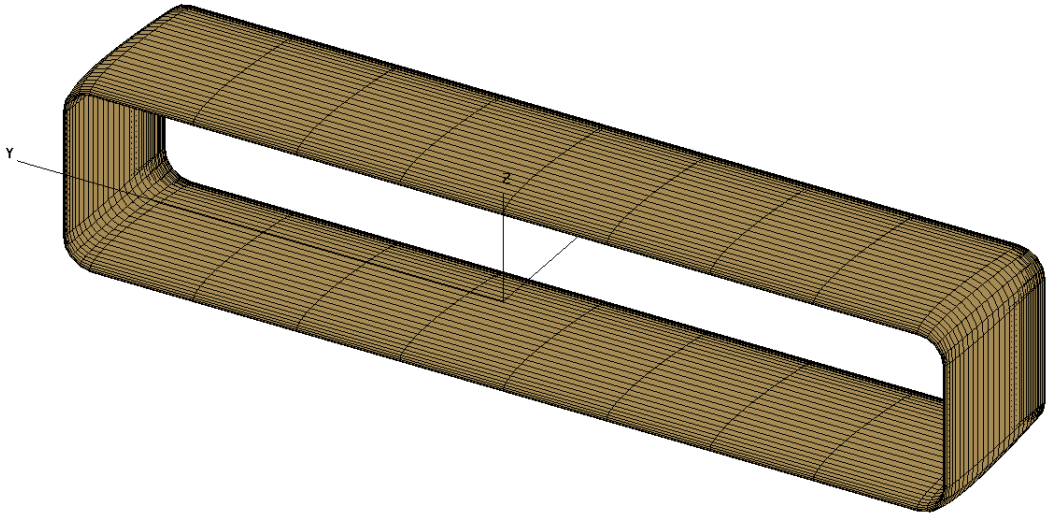


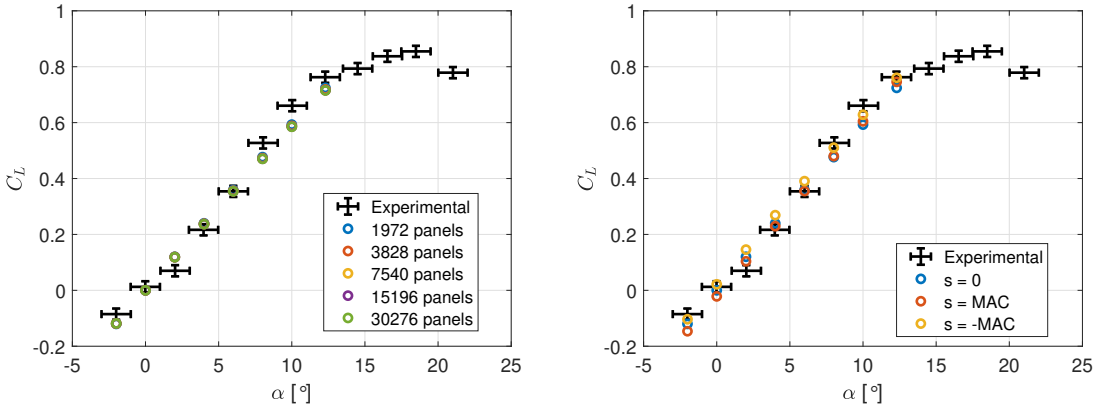
Figure 5.3: Box wing mesh for APAME

different and unknown sign required to repeat this study for the two possible configurations with the lowest mesh resolution. Results are summarized in Table 5.6 and depicted in Figure 5.4b.

APAME software has shown good agreement with experimental results in the linear region of the lift coefficient regardless of the stagger value, with relative errors under 5%. This tool will be used for the definition of a box wing equivalent to a conventional wing in Chapter 7.

N_{bl}	3	10	24	53	110
N_{bu}	3	10	24	53	110
N_{bw}	0	2	6	14	30
N_{af}	59	59	59	59	59
N_R	5	5	5	5	5
Wingpanels	1972	3828	7540	15196	30276

Table 5.4: APAME geometry/mesh resolution setup



(a) C_L vs α curve for different panel resolution

(b) C_L vs α curve for different stagger values

Figure 5.4: APAME mesh resolution

α [°]	**1972 panels**	**3828 panels**	**7540 panels**	**15196 panels**	**30276 panels**
-2	4.0931	3.9807	3.9384	3.9206	3.9073
0	1.4563	1.4572	1.4568	1.4565	1.4566
2	5.8670	5.7535	5.7118	5.6945	5.6810
4	2.7303	2.5042	2.4212	2.3866	2.3595
6	0.5463	0.2110	0.0864	0.0350	0.0052
8	5.9367	6.3792	6.5435	6.6121	6.6651
10	7.9109	8.4571	8.6600	8.7447	8.8104
12.3	4.4671	5.1257	5.3716	5.4744	5.5540
Average	4.1260	4.2336	4.2737	4.2906	4.3049

Table 5.5: Relative error (%) in C_L for different mesh resolutions

α [°]	**s = 0**	**s = MAC**	**s = -MAC**
-2	4.0931	7.1653	2.1506
0	1.4563	3.9915	1.0546
2	5.8670	3.9527	8.8647
4	2.7303	1.5189	6.1298
6	0.5463	0.1186	4.2668
8	5.9367	5.5047	1.9762
10	7.9109	6.5479	3.7886
12.3	4.4671	1.9494	0.2493
Average	4.1260	3.8436	3.5601

Table 5.6: Relative error (%) in C_L for different wing stagger

5.4. CFD MESH

Unlike panel methods, CFD computation require thoroughly generated meshes with high quality. For accuracy reasons it is decided to model the boundary layer (BL) explicitly instead of using wall functions. This is

beneficial for the simulations where boundary layer separation is expected, enhancing the profile prediction of the boundary layer. In terms of AWE systems, continuous high lift operation is desired, being the prediction of the stalling point of the wing key for fulfilling this purpose.

In order to explicitly model the boundary layer, it is necessary to find the thickness of the first layer by imposing a dimensionless wall distance y^+ of 1. Eq 5.3 is used to compute the thickness of the first layer δ_F by knowing the friction velocity u_τ and the kinematic viscosity ν [63].

$$y^+ = \frac{\delta_F u_\tau}{\nu} \rightarrow \delta_F = \frac{y^+ \nu}{u_\tau} \tag{5.3}$$

The friction velocity (u_τ) is computed as

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{5.4}$$

Where ρ is the fluid density and τ_w is the wall shear stress computed as

$$\tau_w = \frac{1}{2} C_f \rho U_\infty^2 \tag{5.5}$$

Where U_∞ is the freestream velocity and C_f the skin friction coefficient. Since the boundary layer is assumed to be turbulent due to the high Reynolds number (5E5<Re_x<1E7), approximation formulas for the local skin friction coefficient of a flat plate are used. In particular, the 1/7 power law (turbulent) is used

$$C_f = 0.0576 Re_x^{-\frac{1}{5}} \tag{5.6}$$

The first layer thickness for the validation geometry is obtained to be δ_F = 8.69E-6 m. Note that this value should be treated as an approximation since it was computed assuming turbulent flow and a flat plate geometry. Further check of the y^+ value in the whole wing is needed to ensure the validity of the results[1].

For the meshing process, the Pointwise tool was used. Pointwise was chosen as the meshing software for several reasons. The first one is the broad record in real aerospace applications, which includes major military aircrafts and spacrafts such as the F-18, F-35 or the International Space Station [38]. The second reason is the flexibility and variety of options for mesh generation the software offers, which makes it suitable for parametric analysis and exploring new concept vehicles. The last reason is the possibility to customize the software by using the API, which opens the possibility for automation maintaining the required quality control specified by the user.

In terms of the meshing process itself, structured meshes are preferred over unstructured meshes for quality reasons. However, since it is not always possible to obtain this type of meshes the hybrid approach was also implemented in this project. Some of the reasons for choosing structured meshes over unstructured ones include [64]:

- Time and memory saving, since for a given volume, less hexahedra (structured) than tetrahedra (unstructured) are required to fill it.
- Resolution, since high quality cells are required for critical regions such as the boundary layer. This is achieved by using high aspect ratio hexahedra.
- Alignment, since more accuracy and easier convergence is expected in the CFD solver when the grid is aligned with the main flow direction. This is an inherent characteristic of structured grids, which are generated following the geomeetry contours.
- Well-defined normals, since they are key for boundary conditions and turbulence modelling. In structured grids the transverse normals are defined easily.

5.4.1. CFD SURFACE MESH

The main spanwise and chordwise divisions for the validation geometry (CAD model) are depicted in Table 5.7. The parameters used to define the surface mesh for the validation geometry are summarized in Table 5.8. Notice these mesh parameters are given per geometrical panel in each of the three regions of the box wing. Figure 5.5 intends to clarify this for the reader.

[1] This check resulted in y^+_{max} = 0.1291, y^+_{min} = 0.0033 and y^+_{avg} = 0.0287 on the wing surface.

N_{bl}	0
N_{bu}	0
N_{bw}	0
N_{af}	82
N_R	10

Table 5.7: Geometrical divisions

Type domain wings	Structured
Type domain winglets	Structured
Type domain curved regions	Structured
Vertical nodes TE	3
Chordwise nodes airfoil	130
Chordwise nodes distribution	Tanh
Chordwise nodes refinement	Element size/2
Spanwise nodes LE/TE wings	151
Spanwise nodes LE/TE winglets	46
Spanwise nodes LE/TE curved regions	10
Spanwise nodes distribution	Uniform
Initial element size curved regions [m]	0.0053
Initial element size wings and winglets [m]	0.0032

Table 5.8: Pointwise surface mesh parameters

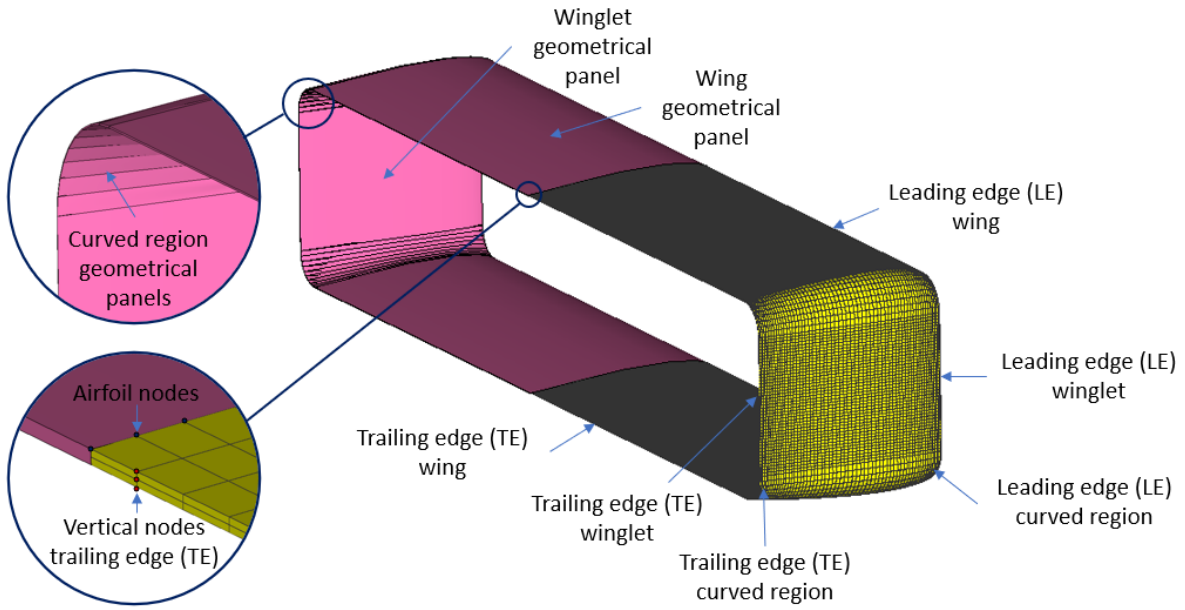


Figure 5.5: Pointwise geometry(pink)/mesh(yellow) difference

In order to generate a proper volume mesh, it is crucial to generate a high quality surface mesh. Before the surface mesh generation, the possibly conflicting regions were identified. In this case, the four wingtips with a 90° bending (upper and lower wings) were identified as hazardous. In particular, the internal part of these corners is crucial for boundary layer extrusion, since cells tend to squeeze instead of expanding, leading to boundary layer mesh collision issues. This difference in cell size when propagating the boundary layer is depicted for a 2D bent in Figure 5.6.

The approach followed in this work consisted of specifying the surface mesh size of the bent region, propagate forward using the growth ratio and geometrical relations and set the last layer size as the initial size of the straight region. Using this approach, more uniformity is ensured in the outer edge of the boundary

layer and the transition towards isotropic cells would be smoother. However, this approach needs to be corrected due to two main reasons. First, the formulas were developed for a planar bent (2D), whereas the bent in this work is three dimensional. Second, the formulas where developed without taking into account airfoil geometry and thus thickness. These two features influence the propagation of the layers, and would make the forward propagation process highly complex. This is the reason why an empirical correction factor was used to account for these 3D effects. Mathematical details of this formulation can be found in Annex A.2. For this particular geometry, the correction factor was set to 0.7.

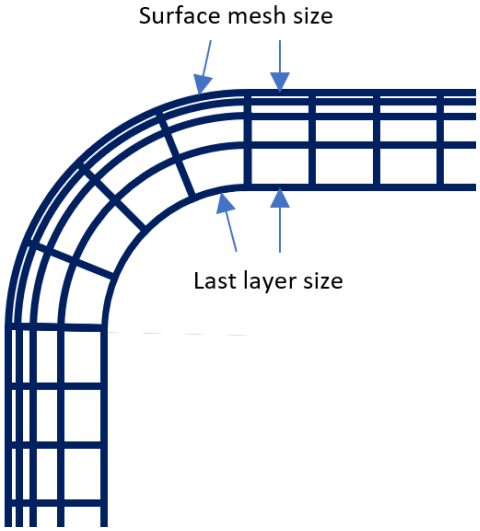


Figure 5.6: Cant radius region

Apart from these conflictive regions, parameters such as the area ratio, the skewness equiangle and the and maximum included angle are monitored[2]. The area ratio is a measure of the relative area of a cell with respect to the adjacent ones. It is mainly useful for finding cell size discontinuities in the surface mesh interfaces. According to guidelines, area ratios of eight to ten in a quad-dominant surface mesh will lead to a volume mesh with high quality [65]. These ratios were identified to be high in the trailing edge of the wing because it was made finite to avoid numerical problems. In particular, they are highly concentrated in the curved regions were the transition form the NACA0012 to NACA0003 occurs. The maximum area ratio in the surface mesh was 24.9781 (Refer to Figure 5.7 for the complete distribution). However, the average area ratio is 1.2886, which is the value that is found in all other regions of the geometry. The skewness equiangle is defined as the maximum ratio between the included angle of the cell and the angle of an equilateral element (60° for triangle, 90° for quad). The variation of this parameter is between 0 (good) and 1 (bad). Following guidelines this quantity should be kept lower than 0.8 for good quality or lower than 0.9 for acceptable quality, depending on the solver [66]. The maximum included angle and skewness equiangle in the surface mesh were 110.7805 and 0.2329 respectively. Since both measures are equivalent, only the skewness equiangle distribution is shown for illustrative purposes in Figure 5.8.

5.4.2. CFD VOLUME MESH

Once the surface mesh quality is within acceptable limits, the volume mesh quality criteria shall be checked. In particular, parameters such as cell non-orthogonality, skewness centroid, skewness equiangle, maximum included angle, volume ratios and boundary layer aspect ratio will be checked.

Cell non-orthogonality is a measure of the angle between two vectors (Refer to Figure 5.9). The first one is the line connecting two neighbouring cell centers across their shared face. The second one is the vector normal to the cell face [67]. Ideally, all cells would be generated properly and non-orthogonality would not exist. However, reality is that numerical meshes are rarely orthogonal, and strategies for a high quality volume mesh are focused on reducing the number of severely orthogonal cells ($> 70°$). In terms of the solvers,

[2]The validation case with low resolution (27811524 cells) at $\alpha = 4°$ was chosen to show representative values of the quality metrics, all other cases present similar metrics and will not be presented to avoid repetition.

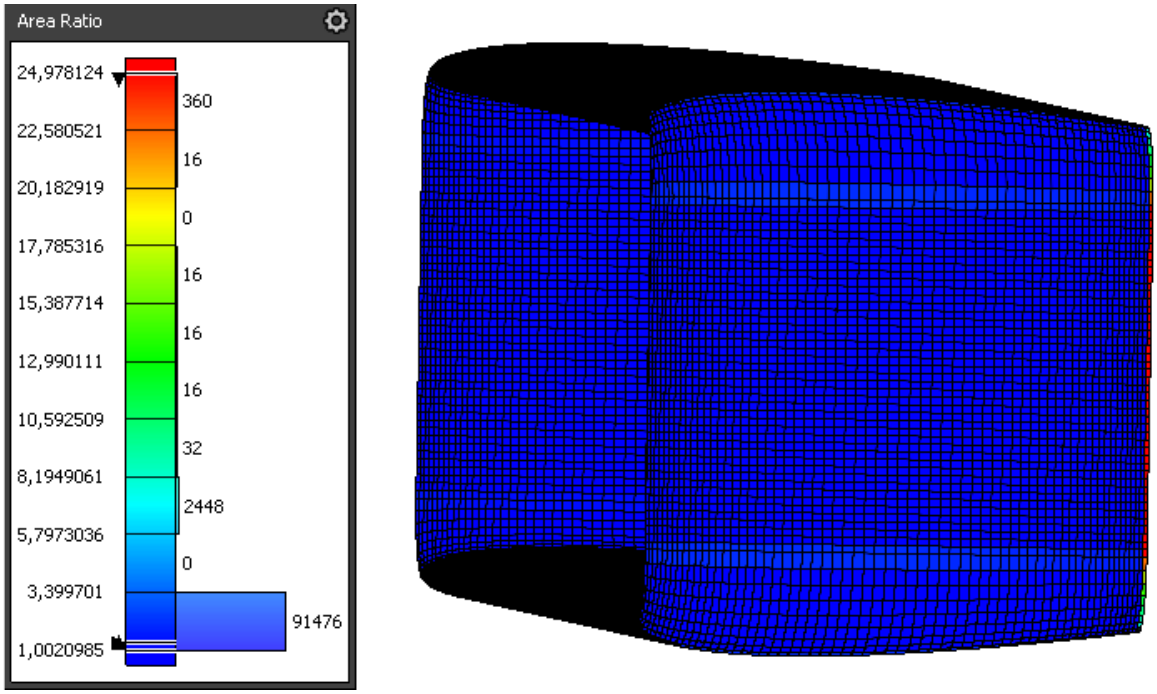


Figure 5.7: Pointwise distribution of the area ratio

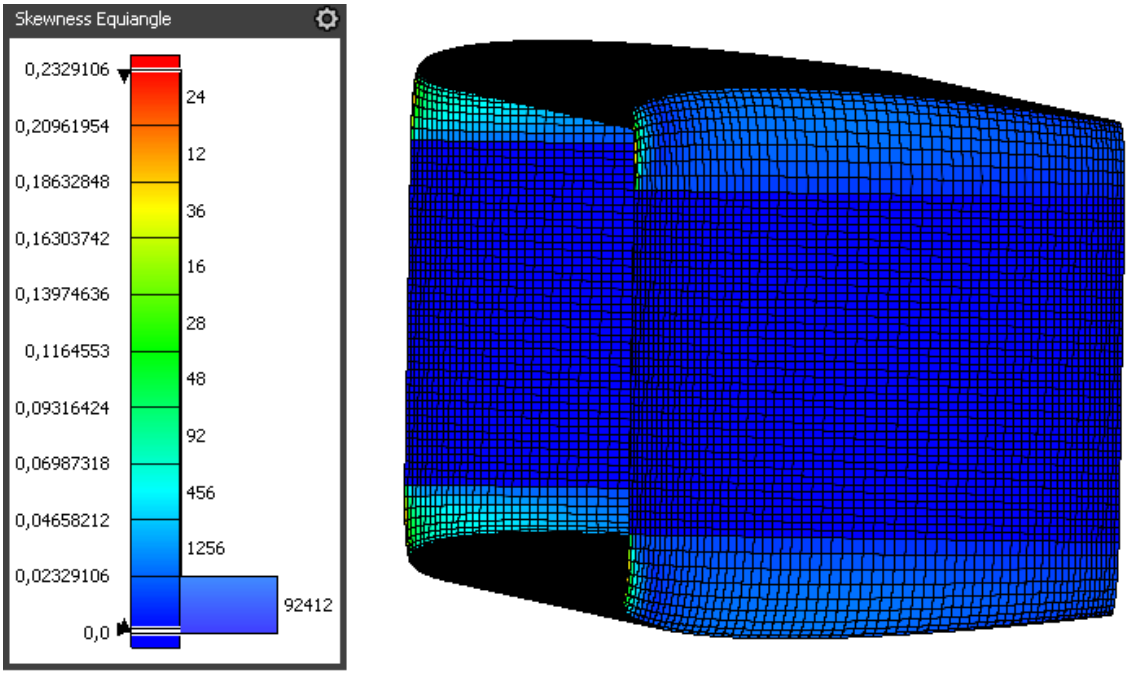


Figure 5.8: Pointwise distribution of the skewness equiangle

non-orthogonal correctors are used when the mesh has some relevant degree of non orthogonality (> 5°). However, if non-orthogonal cells are above 80°, the mesh will be automatically discarded since convergence will be really difficult to achieve [68].

Centroid skewness is defined as 1 minus the minimum dot product between two vectors. The first one is the one connecting both cell and face centroid. The second one is the face normal vector. Values of this metric range from 0 (zero skew) to 1 (collapsed cell) [69].

Both skewness equiangle and maximum included angle follow similar definition and guidelines as the ones stated in the previous section. Volume ratio is a measure of the relative volume of a cell with respect to

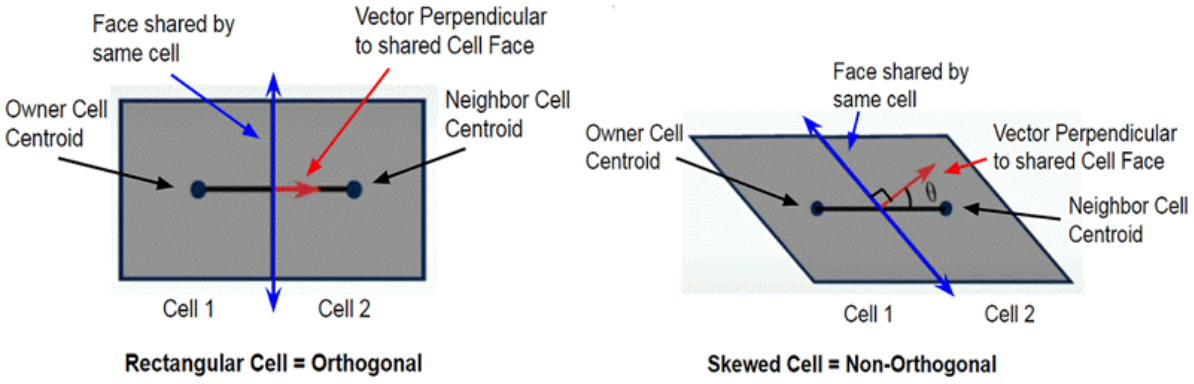


Figure 5.9: Cell non-orthogonality [67]

5

the adjacent ones. Since it is equivalent to the area ratio in a surface mesh, similar guidelines are followed (between eight to ten).

Finally, the aspect ratio is defined depending on the cell type. For hexahedral cells, the aspect ratio is calculated as the ratio between the maximum and the minimum of the length, width and height, being 1 the aspect ratio of a cube. For tetrahedral cells, it is computed as the ratio between the radius of the sphere circumscribing the cell and the radius of the inscribed sphere times three. For prism cells, it is the ratio between the prism average height and the average length of the edges belonging to the triangular base. For pyramid cells, it is the ratio between the pyramid height and the average length of the edges belonging to the quadrilateral base [70]. Note that these values are expected to be large in the first layers of the boundary layer (of the order of 10^2-10^3).

Figure 5.10 summarizes the values of the previously described metrics. Again since volume ratio is equivalent to area ratio, values at the trailing edge are expected to be high. However, the average is 1.6807, which mean the majority of the regions are well below the recommended thresholds.

For the generation of the volume mesh itself, the 3D anisotropic tetrahedral extrusion (T-rex) tool within Pointwise was used. This tool allows a complete control over the boundary layer and the isotropic cells generation [71]. For the boundary layer, anisotropic layers containing tetrahedra, prisms, hexahedra and pyramids are allowed. However, it would be desirable to have structured hexahedra for the boundary layer instead of unstructured tetrahedra. This is convenient for two reasons: the first one comes directly from the surface mesh being structured, leading to also structured extrusion of the boundary layer. The second one, is related to a Pointwise option called 'Convert Wall Domains' which turns triangular cells associated to the aerodynamic body into quad dominant cells. Tetrahedral cells are used for the isotropic cells in the farfield where quality is less important than in the boundary layer. However, some refinements are done in the regions of interest outside of the boundary layer in order not to lose the information gained by the high quality cells in the boundary layer. They are discussed in the next section.

Parameters defining the volumetric mesh generation are summarized in Table 5.9. Figure 5.11 shows the hexahedral cells present in the boundary layer extrusion in sections y = 0, x = MAC/2 and z = h/2. Figure 5.12 depicts the variety of cells present in the volume mesh: in blue, hexahedra extruded for the boundary layer are shown; in yellow, pyramids aiming to smooth the transition between hexahedra and tetrahedra are presented; and in red, refined tetrahedra present in the isotropic region (farfield) are displayed.

Max layers BL	34
Full layers BL	34
Growth rate BL	1.2
Solver attribute	AllAndConvertWallDoms
First layer height BL [m]	8.69e-6
Max growth rate isotropic cells	1.2

Table 5.9: Pointwise volume mesh parameters

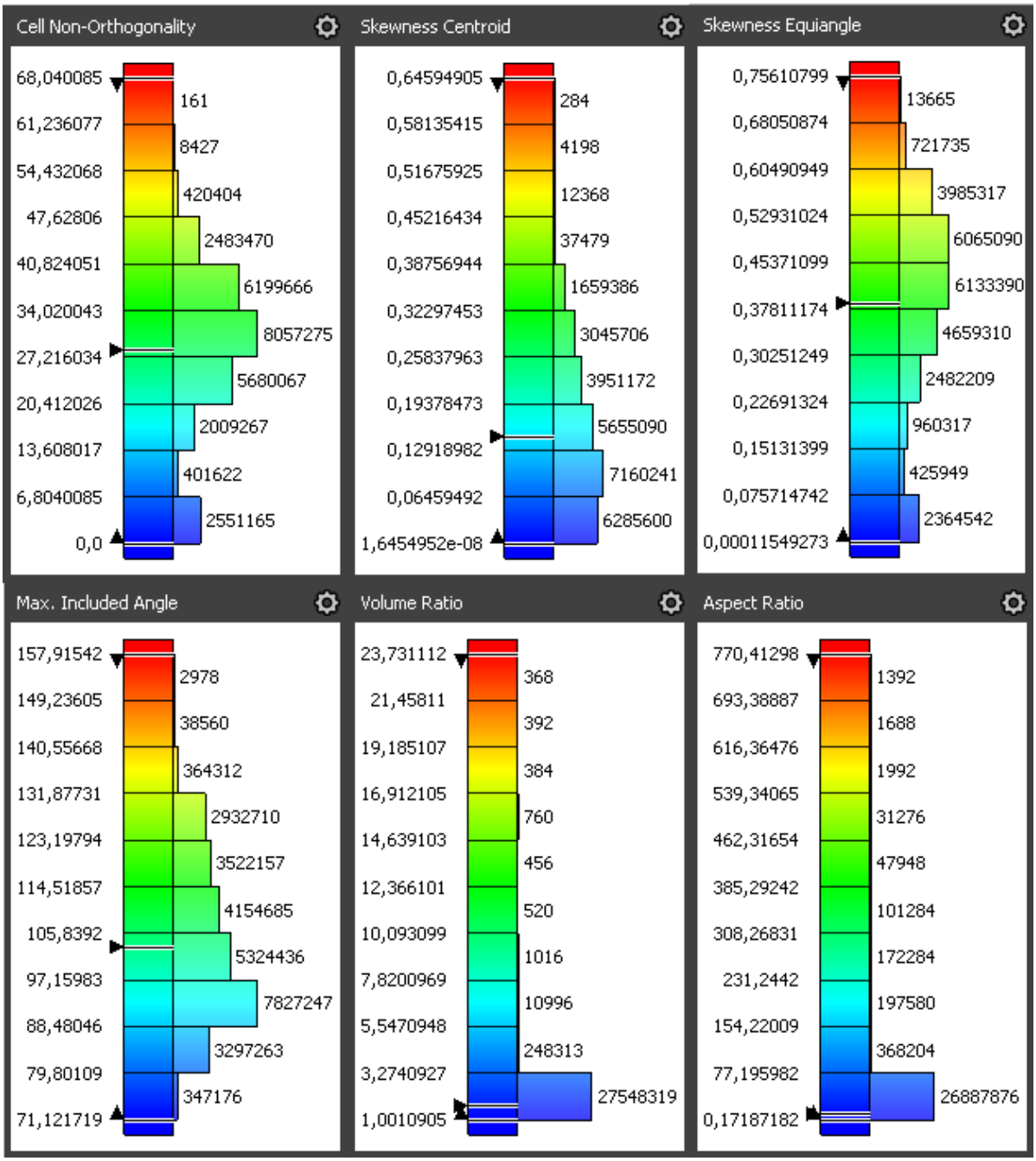


Figure 5.10: Pointwise volume mesh quality metrics

5.5. CFD MESH RESOLUTION STUDY

A key goal in CFD is to achieve the highest reliability with the lowest computational resources. In this case, a mesh sensitivity study is performed to check the magnitude of the error made in C_L and C_D when compared to experimental data. Three different resolutions are tested for different angles of attack in this work, all of them with the same farfield, but using different refinement regions.

In terms of the farfield dimensions, there is not a universal consensus of which should be the extent. It was found by Athadkar & Desai [72] that for their 2D airfoil simulation using Spallart-Allmaras turbulent model, a farfield extending 10 chords upstream and 15 chords downstream provided acceptable results. To be conservative and avoid the boundary conditions influencing the flow over the wing, in the validation cases the box farfield extends 15 chords upstream, 40 chords downstream and 15 chords in each of the other directions (top, bottom, port and starboard).

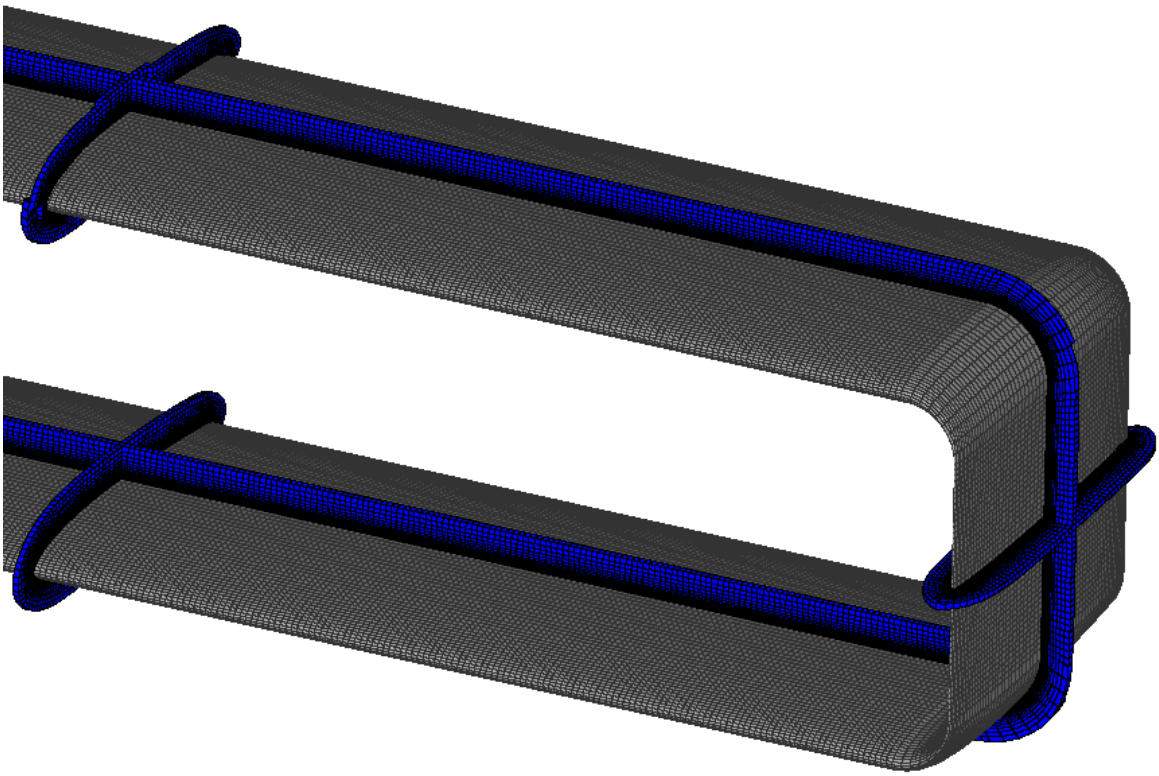


Figure 5.11: Pointwise volume mesh boundary layer

In terms of the refined regions, they are depicted in Figure 5.13 and their parameters are found in Table 5.10. The first one is a box enclosing the geometry and extending 25% more in each direction: upstream, downstream, top, bottom, port and starboard. This is considered the base case since this refinement is only relevant for the boundary layer transition from structured to isotropic cells in the gap between the two wings. It is also important for the transition in the external part of the wing. However, the main limitation is that the wake is not captured properly due to the short downstream length of this refined region. Notice that the refinement box rotates together with the rotation associated to the geometry due to a variation in angle of attack.

The second and third refinements aim to capture the wake properly. Since it is difficult to predict the direction of the wake leaving the trailing edge, it was observed in initial simulations that the wake leaves in a direction half the angle of attack, which in this case is the average direction between the trailing edge and the direction of the freestream. The extent of these refinement regions are 10 chords and 20 chords downstream, respectively.

Attribute	**Refinement 1**	**Refinement 2**	**Refinement 3**
Type	Constant	Parametric	Parametric
Initial spacing [m]	0.004	0.0045	0.0045
Initial decay	0.3	0.3	0.3
Final spacing [m]	-	0.015	0.015
Final decay	-	0.3	0.3

Table 5.10: Pointwise refinement parameters

5.6. CFD SIMULATION SETUP

The solver used for the RANS simulations is OpenFOAM, which is an open-source leading software in computational fluid dynamic applications [39]. The version used is the one released in June 2020 (v2006), which

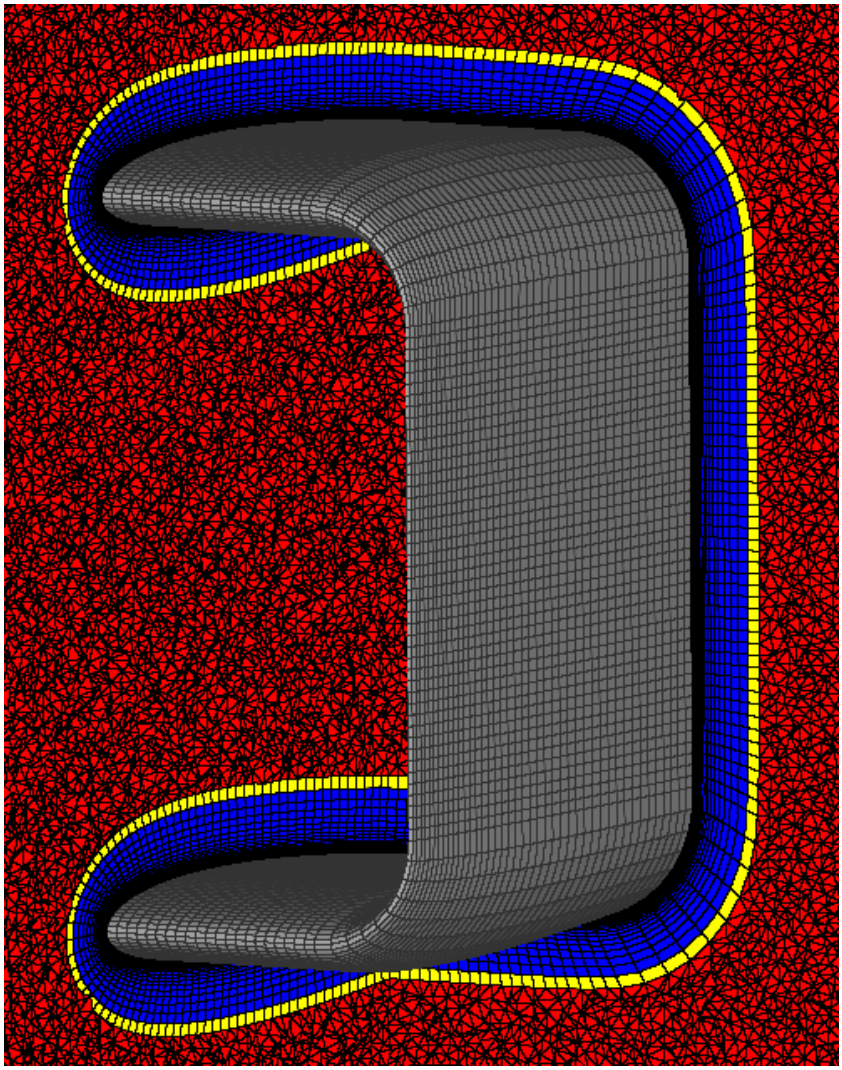


Figure 5.12: Pointwise volume mesh: hexahedra (blue), pyramids (yellow) and tetrahedra (red)

is the last version installed in the high performance cluster where simulations are performed. The data obtained from experiments contain both lift and drag coefficients, which are computed in OpenFOAM following Eqs. 5.7 & 5.8.

$$C_L = \frac{L}{\frac{1}{2}\rho_\infty U_\infty^2 S_{ref}} \tag{5.7}$$

$$C_D = \frac{D}{\frac{1}{2}\rho_\infty U_\infty^2 S_{ref}} \tag{5.8}$$

With L and D, the lift and drag force, ρ_∞ the freestream density, U_∞ the freestream velocity and S_{ref} the reference area. This reference area is the planform area, which in this case needs to take into account both lifting surfaces S_u and S_l (upper and lower wing) as described in Eq. 5.9.

$$S_{ref} = S_l + S_u = 2\int_0^{b/2} c_l(y)\,dy + 2\int_0^{b/2} c_u(y)\,dy \tag{5.9}$$

With c_u and c_l the local chord of the upper and lower wings respectively.

In order to exploit the parallelisation capabilities of OpenFOAM and considering the amount of cells in the CFD mesh it was decided to decompose it into several sub-domains. Each sub-domain is solved in a different processor (CPU), which can communicate between each other and are joined back together when the simulation is finished. The configuration file in charge of splitting the full domain into sub-domains

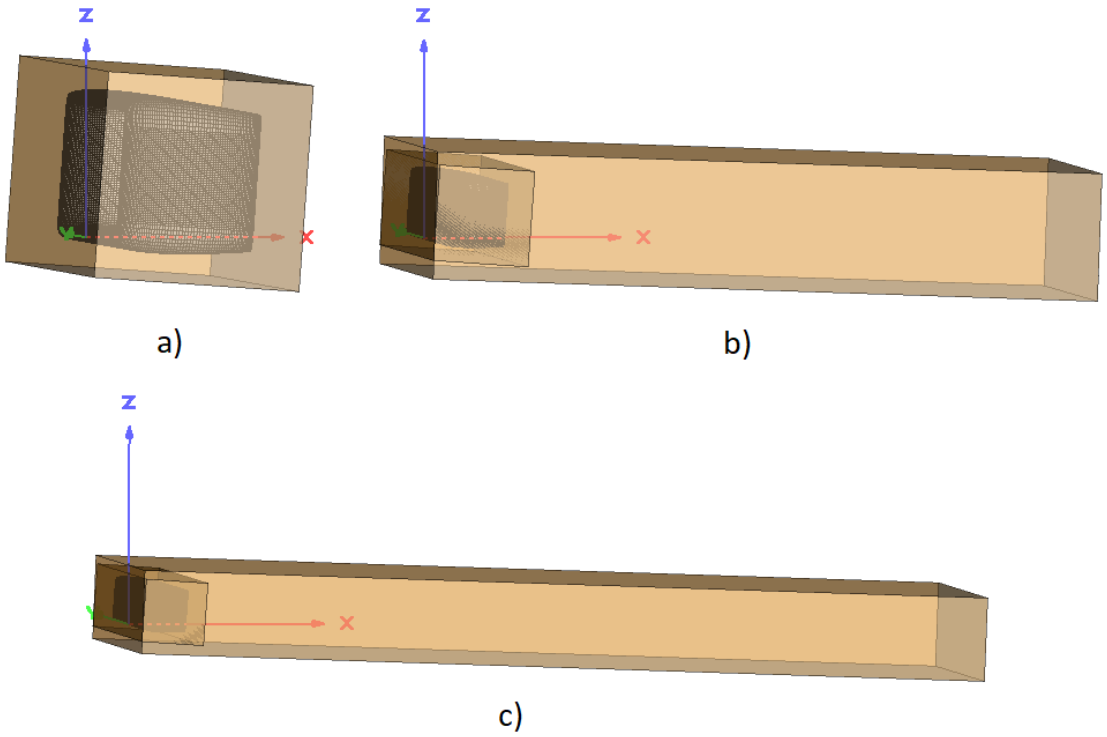


Figure 5.13: Pointwise mesh refinement cases: a) Low resolution case ≈ 28M cells, b) Medium resolution case ≈ 51M cells and c) High resolution ≈ 83M cells

given a decomposition method and number of cores is named `decomposeParDict`. Scotch decomposition algorithm was chosen because it requires no input from the user and automatically tries to minimise the number of CPU boundaries [73]. All simulations but test cases in this work were run using a single node and 48 cores.

5.6.1. INITIAL VALUES AND BOUNDARY CONDITIONS

The initial and boundary condition values used in the simulations were defined in a separate file named `includeDict`. The first of them is the initial velocity vector $\vec{U}_i$, which is calculated specifying the angle of attack α, the angle of sideslip β and the magnitude of the velocity U_∞ as in Eq. 5.10. i, j and k are the chordwise, spanwise and upwards projections respectively. However, this initial velocity vector is always taking α = 0° and β = 0°, since the rotation for different angles of attack or sideslip occurs in the meshing process and not in the boundary conditions itself. Note that rotating the geometry is the approach followed when performing wind tunnel experiments and it is convenient to replicate the same setup and flight conditions when validating the CFD simulations.

$$\vec{U}_i = U_\infty \cdot (i, j, k) = U_\infty \cdot (\cos(\beta)\cos(\alpha), -\sin(\beta), \sin(\alpha)) \tag{5.10}$$

The second quantity is the Reynolds number, defined by Eq. 5.11. Where μ_∞ is the dynamic viscosity of the freestream and l_c the characteristic length of the body. This characteristic length in this work is set to be the mean aerodynamic chord (MAC) taking into account both lifting surfaces MAC_l and MAC_u (upper and lower wing) which is computed according to Eq 5.12.

$$Re = \frac{\rho_\infty U_\infty l_c}{\mu_\infty} \tag{5.11}$$

$$\begin{aligned} MAC &= MAC_l \frac{S_l}{S_{ref}} + MAC_u \frac{S_u}{S_{ref}} = \\ &= \left(\frac{2}{S_l}\int_0^{b/2} c_l(y)^2\, dy\right)\frac{S_l}{S_{ref}} + \left(\frac{2}{S_u}\int_0^{b/2} c_u(y)^2\, dy\right)\frac{S_u}{S_{ref}} \end{aligned} \tag{5.12}$$

The third quantity to be initialized is the turbulent kinetic energy of the eddies in the flow, denoted as k_i. According to Eq. 5.13 they are dependent on the initial freestream speed U_∞, and the turbulence intensity T_I which is given in percentage.

$$k_i = \frac{3}{2}\left(\frac{U_i T_I}{100}\right)^2 \tag{5.13}$$

The turbulence intensity is classified into high, medium and low turbulence cases. High turbulence typically presents values between 5-20%, medium turbulence between 5-1% and low turbulence below 1%. For external aerodynamic applications, the turbulence intensity is usually low [74]. Since the quality of the wind tunnel used for the experimental data is unknown, it was assumed a turbulence intensity of 2% for the experiments.

The fourth to be computed during the initialization is the specific turbulent dissipation rate ω_i. Two possibilities are found for the initialization of this variable: from the turbulent length scale or from the eddy viscosity ratio. The second approach was followed given that it is suitable for external aerodynamics, whereas the first one is more focused on internal flow applications [75]. According to Eq. 5.14 this quantity is dependent on the turbulent kinetic energy k_i, the kinematic viscosity of the freestream ν_∞ and the eddy viscosity ratio $\frac{\nu_t}{\nu_\infty}$.

$$\omega_i = \frac{k_i}{\nu_\infty}\left(\frac{\nu_t}{\nu_\infty}\right)^{-1} \tag{5.14}$$

The fift and last quantity to be initialized is the ratio between the freestream pressure and the freestream density p_i/ρ_i. Incompressible simulations in OpenFOAM do not use the pressure but operate directly with the pressure over density ratio. CFD simulations for AWE applications do not exceed M = 0.3, so it is safe to assume incompressible flow. Table 5.11 summarizes the type of boundary condition used in the different regions together with the initial and boundary values.

Boundary quantity	**Farfield type**	**Farfield value**	**Wall type**	**Wall value**
$U\,[m/s]$	`inletOutlet`	`inletValue` = U_i Initial value = 0	`fixedValue`	(0 0 0)
$p\,[m^2/s^2]$	`outletInlet`	`outletValue` = p_i/ρ_i Initial value = p_i/ρ_i	`zeroGradient`	-
$\omega\,[1/s]$	`inletOutlet`	`inletValue` = ω_i Initial value = ω_i	`omegaWallFunction`	Initial value = ω_i
$\nu_t\,[m^2/s]$	`calculated`	Initial value = 0	`nutkWallFunction`	Initial value = 0
$k\,[m^2/s^2]$	`inletOutlet`	`inletValue` = k_i Initial value = k_i	`fixedValue`	Value = 0

Table 5.11: Initial and boundary conditions definition

5.6.2. NUMERICAL SCHEMES AND SOLVER SETTINGS

Any numerical simulation requires two conditions. The first one is stability on the numerical solution, which means errors remain bounded when the iteration process advances. The second one is consistency in the numerical scheme, which means the numerical scheme must tend to the differential equation when the time and spatial steps are infinitesimally small. If these two conditions are fulfilled then convergence is eventually fulfilled. In the steady state simulations, it was decided to keep a fixed amount of iterations instead of specifying a stopping convergence criteria to see the evolution of the forces and the residuals at different angles of attack and mesh resolutions. However, Cauchy convergence criterion was used to judge if the number of fixed iterations chosen was giving converged values. This criterion is formulated as follows:

$$\epsilon_c = \frac{1}{N_{it}-1}\sum_{n=0}^{N_{it}-1}\left|X_{N_{sim}-n} - X_{N_{sim}-n-1}\right| \tag{5.15}$$

With X the variable of interest, N_{sim} the fixed number of iterations ran in the simulation, N_{it} the number of iterations taken into account for the calculation of the error starting from the last value and propagating backwards. ϵ_c is a measure of the average error between each step in the last N_{it} iterations of the simulation. The quantities of interest in this work are C_L and C_D. Since these quantities are already normalized, an acceptable value for convergence was assumed to be $\epsilon_c <= 10E-4$ with a $N_{it} = 1000$, which implies a variation on these quantities in the fourth decimal place.

It is not the purpose of this work to provide a deep study in OpenFOAM simulation settings. For this reason, most of the settings were chosen to be based on the ones used by Lebesque [76], who performed RANS steady state simulations on a leading edge inflatable kite (LEI) imposing rigid geometry for $Re \sim$ 1E6.

The files that contain all the information regarding numerical schemes and solver settings are `fvSchemes` and `fvSolution` respectively. The main categories [77] and the chosen numerical schemes for steady state simulations are summarized in Table 5.12. The time scheme is set to steady state which means temporal derivatives are zero and timestep is non relevant for the solution. Gradient schemes are set to Gauss linear, which interpolates the values of the cell centres to the face centres by using central difference. Divergence schemes are chosen to be bounded upwind for stability reasons. For advective terms like the flow velocity U, the chosen scheme is bounded Gauss linear Upwind default (second order), whereas for non-advective terms like k or ω bounded Gauss linear (first order) is enough. Laplacian scheme (diffusion) is chosen to be Gauss linear corrected to account for cell non-orthogonality. Surface normal gradients are also corrected for the same reason.

Category	**Chosen scheme**
Time	`steadyState`
Gradient	`Gauss linear`
Divergence (U)	`bounded Gauss linearUpwind default`
Divergence (k,ω)	`bounded Gauss upwind`
Laplacian	`Gauss linear corrected`
Interpolation	`linear`
Surface normal gradient	`corrected`

Table 5.12: Numerical schemes used

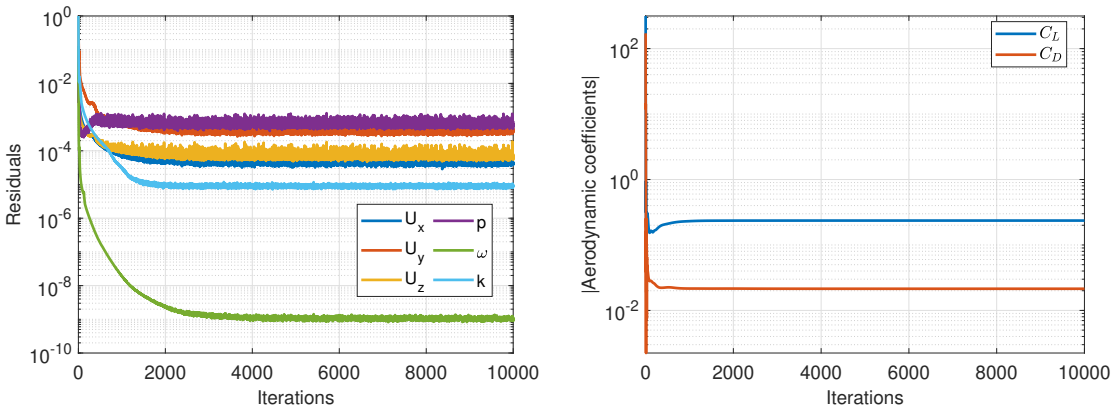
The solver settings associated to the different categories are summarized in 5.13. For the pressure term, the Geometric Algebraic Multi Grid (GAMG) solver is used together with a Gauss seidel smoother. The working principle is to use a coarse grid/matrix for a fast initial solution that is used to smoothen high frequency errors and as starting point for the finer grid/matrix. For the other variables, an iterative solver (smooth solver) using Gauss-Seidel smoother is used. SIMPLE settings refer to the SIMPLE algorithm explained in Chapter 4. In this case the SIMPLE consistent (SIMPLEC) algorithm is used because of a better rate of convergence (faster). This algorithm follows the same steps as SIMPLE except for a small manipulation of the momentum equations that makes SIMPLEC velocity correction equations to discard terms that are less relevant than the velocity neighbour correction terms discarded in SIMPLE [62]. According to the mesh characteristics of the validation geometry, there are no severely orthogonal cells ($> 70°$), but a small percentage close to that value. This is the reason why the number of orthogonal correctors is set to 1. In terms of relaxation factors, they were set to 0.3 for the velocity and 0.5 for all other variables, except pressure. This increased the stability of the calculations at the expense of decreasing the rate of convergence. Finally, at the beginning of steady state simulations, ten non orthogonal potential flow corrector steps are set for flow field initialization.

Category	**Settings**
Solver (p)	`GAMG` smoother `GaussSeidel`
Solver (U,k,ω)	`smoothSolver` smoother `symGaussSeidel`
SIMPLE	consistent `yes` nNonOrthogonalCorrectors 1
Relaxation factors U k,ω	 0.3 0.5
potentialFlow	nNonOrthogonalCorrectors 10

Table 5.13: Solver control settings used

5.6.3. MONITORING RESIDUALS AND MESH RESOLUTION STUDY

All simulations were run for 10000 iterations and the residuals from the fluid variables as well as the aerodynamic coefficients magnitude were monitored. Figure 5.14 is an illustrative example of the monitoring process for the $\alpha = 4°$ low resolution simulation. Once the simulation is finished, convergence is evaluated using Cauchy convergence criterion for the last 1000 iterations (Refer to Eq. 5.15). This criterion is used to check whether the simulation would need to continue for more iterations or if a stable solution has been reached within the specified iterations. Tables 5.14 & 5.15 summarize the Cauchy measured error for all cases. In this case, all of them were below 10^{-4}, which means maximum variations in the fourth decimal place of the C_L and C_D are expected in the last iterations. Two conclusions can be extracted from the values depicted in these tables. The first one is that higher resolutions, on average, present higher values of the Cauchy metric, which means convergence would need more iterations to be equal to the metrics obtained for lower resolutions. The second one is that higher angles of attack, in general, take more time to converge than lower angles of attack. This is explained by the increased complexity that the non linear region has in the simulations at high angles of attack. Effects such as boundary layer separation and stall are expected in these regions.



(a) Residual fluid variables

(b) Aerodynamic coefficients

Figure 5.14: Monitoring process

α [°]	Low resolution	Medium resolution	High resolution
-2	3.9386E-6	3.878E-6	5.0665E-6
4	8.9057E-6	1.1465E-5	1.6577E-5
12.3	2.4136E-6	3.0244E-5	3.4126E-5
14.5	3.1201E-5	3.5657E-5	8.4527E-5
16.5	3.2133E-5	3.8688E-5	7.5248E-5
18.5	4.8208E-5	7.0399E-5	4.7046E-5
21	1.9413E-5	3.9170E-5	6.7688E-5
Average	2.3991E-5	3.2786E-5	4.7182E-5

Table 5.14: Cauchy convergence on C_L

Note that even if the simulation has converged for a particular mesh resolution, this does not imply the solution is accurate. Accuracy of the solution was evaluated using the numerical results on the aerodynamic coefficients of the three different resolutions. In particular, Richardson extrapolation [78] is used for a complete grid convergence analysis. The main goal of this procedure is to obtain solutions for different grid levels and then extrapolate them to the case Δx=0. Note that this case would correspond to having the exact solution of the variables of interest. In this work, these variables are the lift and drag coefficients. For the case of three grid levels, Richardson extrapolation is performed solving for the unknowns f_{exact}, B and i, numerically in the system of equations shown in Eq 5.16.

α [°]	Low resolution	Medium resolution	High resolution
-2	3.6315E-7	3.9621E-7	4.8546E-7
4	4.5728E-7	1.3762E-6	8.8671E-7
12.3	4.1677E-6	3.7196E-6	5.3443E-6
14.5	5.3966E-6	7.6190E-6	2.1552E-5
16.5	7.8101E-6	8.8307E-6	1.9061E-5
18.5	1.2261E-5	2.3930E-5	1.8713E-5
21	8.5350E-6	8.8107E-6	2.0387E-5
Average	5.5701E-6	7.8117E-6	1.2347E-5

Table 5.15: Cauchy convergence on C_D

$$\begin{cases} f_1 = f_{exact} + B\Delta x_1^i \\ f_2 = f_{exact} + B\Delta x_2^i \\ f_3 = f_{exact} + B\Delta x_3^i \end{cases} \tag{5.16}$$

Where f denotes the variable of interest, B a coefficient for the curve, i the apparent order of the method and Δx the grid level. Values of C_L and C_D for the different mesh resolutions and angles of attack, together with the value obtained using Richardson extrapolation are shown in Tables 5.16 & 5.17. Absolute relative errors with respect to the estimated exact value using Richardson extrapolation are also depicted in these tables. The first feature extrated from this process is that using a low resolution for values of the angle of attack where non linear effects are not predominant, is enough to obtain a value close (< 0.1%) to the estimation of the exact value of C_L. However, for the point of maximum lift coefficient ($\alpha = 18.5°$) and stall ($\alpha = 21°$), relative absolute errors exceed 10% for lower resolution creating a significant difference to the exact value. A similar behaviour applies for the drag coefficient. However, even if the errors show lower values, there is a drag contribution that comes directly from the lift (induced drag). This contribution is also affected by the error when estimating the lift coefficient, being this quantity the main driver when choosing a mesh resolution. In this case, medium or high resolution are proven acceptable for the maximum lift point, whereas for analyzing stall, a high resolution would be preferred.

α [°]	Low resolution	Medium resolution	High resolution	Richardson
-2	-0.1182 (0.25%)	-0.1183 (0.12%)	-0.1184 (0.05%)	-0.1185
4	0.2360 (0.16%)	0.2362 (0.08%)	0.2363 (0.04%)	0.2364
12.3	0.7021 (0.19%)	0.7030 (0.06%)	0.7032 (0.03%)	0.7034
14.5	0.8098 (0.11%)	0.8109 (0.03%)	0.8108 (0.01%)	0.8107
16.5	0.8931 (0.09%)	0.8938 (0.01%)	0.8938 (0.005%)	0.8939
18.5	0.8759 (10.36%)	0.9453 (3.26%)	0.9631 (1.44%)	0.9771
21	0.6397 (12.69%)	0.6863 (6.34%)	0.7122 (2.80%)	0.7327

Table 5.16: Numerical values for C_L

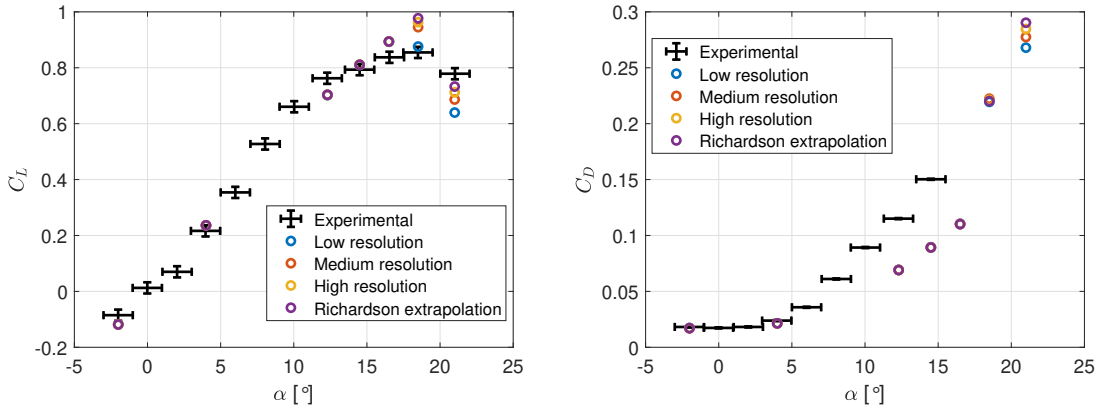
α [°]	Low resolution	Medium resolution	High resolution	Richardson
-2	0.01715 (0.056%)	0.01708 (0.002%)	0.01708 (0.001%)	0.01708
4	0.02144 (0.036%)	0.02137 (0.003%)	0.02136 (0.002%)	0.02136
12.3	0.06926 (0.029%)	0.06914 (0.012%)	0.06909 (0.005%)	0.06905
14.5	0.08945 (0.024%)	0.08934 (0.011%)	0.08930 (0.005%)	0.08926
16.5	0.11043 (0.036%)	0.11025 (0.016%)	0.11017 (0.007%)	0.11011
18.5	0.21950 (0.075%)	0.22235 (0.217%)	0.22116 (0.096%)	0.22023
21	0.26800 (3.042%)	0.27752 (1.743%)	0.28464 (0.770%)	0.29029

Table 5.17: Numerical values for C_D

5.6.4. COMPARISON TO EXPERIMENTAL DATA

Having performed the mesh resolution study and estimated the exact value that could be obtained from the simulations, a comparison with the experimental data used for validation is performed. Figure 5.15 shows the data obtained with CFD simulations and their relative position to the experimental data. In general, the trend in lift coefficient is in good agreement with the experiments except for the maximum lift coefficient point. In terms of drag coefficient, CFD results are aligned with those obtained from experiments for low angles of attack. The causes of these differences are identified to be mainly two. The first one, is due to the stagger between wings. As previously mentioned, although the box wing from the experiments presents stagger, it was decided to perform a zero stagger case for the CFD simulations due to the unknown sign of the stagger and time constraints. This difference is thought to affect mostly the drag coefficient, but also the lift coefficient at high angles of attack. Having some stagger between wings changes the effective angle of incidence facing each wing, provided that the vertical separation between them allows for flow interaction. It also changes the wing area facing the flow at angles of attack different than zero, changing this way the form drag. The second reason contributing to differences between CFD and experiments is the cant radius. The value of this quantity was not specified in the experiments, and the assumed value may induce some differences in lift and drag coefficients.

Quantification of this difference, together with the value of the coefficients is presented in Tables 5.18 & 5.19. Relative error was computed in this case as defined in Eq. 5.2. As previously mentioned, errors in C_L do not exceed 7% except on the maximum lift coefficient point. However, relevant differences in the drag coefficient are found for $\alpha = 12.3°$ and $\alpha = 14.5°$.



(a) C_L vs α curve

(b) C_D vs α curve

Figure 5.15: Comparison between CFD and experiments

α [°]	Richardson	Experimental
-2	-0.1185 (3.90%)	-0.0851
4	0.2364 (2.32%)	0.2165
12.3	0.7034 (6.91%)	0.7625
14.5	0.8107 (2.04%)	0.7933
16.5	0.8939 (6.61%)	0.8374
18.5	0.9771 (14.31%)	0.8549
21	0.7327 (5.38%)	0.7787

Table 5.18: Relative error (%) using CFD on C_L

α [°]	**Richardson**	**Experimental**
-2	0.01708 (0.732%)	0.01818
4	0.02136 (1.668%)	0.02386
12.3	0.06909 (30.611%)	0.11568
14.5	0.08930 (40.607%)	0.15028

Table 5.19: Relative error (%) using CFD on C_D

6

POST-PROCESSING

The goal of this chapter is to present and post process the numerical results obtained from the simulations. It begins with a brief description of the quantities to be analyzed followed by a presentation of this quantities for different flight conditions. Finally, the main trends are discussed in the last section of this chapter.

6.1. POST-PROCESSING QUANTITIES

In this section, the relevant quantities used for the post process are defined. The first one is the pressure coefficient [79], which measures the relative pressures acting on the flowfield in a non dimensional form. It is defined in Eq. 6.1, where p is the pressure of a particular point, S_{ref} the reference lifting surface and freestream pressure, density and velocity magnitude denoted with p_∞, ρ_∞ and U_∞, respectively. This equation can be further simplified for incompressible flows using Bernoulli's equation to Eq. 6.2, where U is the magnitude of the velocity at a particular point.

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty U_\infty^2} \tag{6.1}$$

$$C_p = 1 - \left(\frac{U}{U_\infty}\right)^2 \tag{6.2}$$

The simplified form of the C_p leads to four different trends depending on the sign and value of this coefficient. If $C_p = 0$ it means the velocity magnitude at a particular location matches that of the freestream, which means pressure is matched as well to the freestream value. If $C_p > 0$ the particular point experiences higher pressure values and lower speeds than the freestream up to reaching zero velocity point. If $C_p = 1$, there is an stagnation point ($U = 0$) at the particular location, with the pressure value being equal to the stagnation pressure. Note that the maximum value C_p can assume is 1. The last trend occurs if $C_p < 0$, the flow will experience lower pressure and higher speeds than the freestream. Note that in this cases C_p values can be lower than -1, meaning a higher speed on these regions.

The second quantity to be analyzed is flow vorticity $\vec{\omega}$ on the plane perpendicular to the aerodynamic geometry chordwise direction. This quantity is related to the spinning motion present at a particular point of the flow. It is computed following Eq. 6.3 [80].

$$\vec{\omega} = \nabla \times \vec{U} \tag{6.3}$$

Recap that in this case the geometry was rotating when the angle of attack changed, leaving the boundary conditions intact. A generic 2D cut in the spanwise direction is represented in Figure 6.1 to show the normal vector to the previously mentioned plane, $\vec{n}$. In this image, x_{af} is the axis following the chordwise direction of the airfoils and z_{af} is the axis connecting the lower and upper airfoils perpendicular to x_{af} pointing upwards.

The vorticity component to be analyzed is computed depending of the angle of attack following Eq. 6.4.

$$\omega_n = \omega_x \cos\alpha - \omega_z \sin\alpha \tag{6.4}$$

Finally, last quantity to be analyzed will be the velocity parallel to the freestream, which in this case is U_x. This is presented together with the streamlines defined by the vector velocity field.

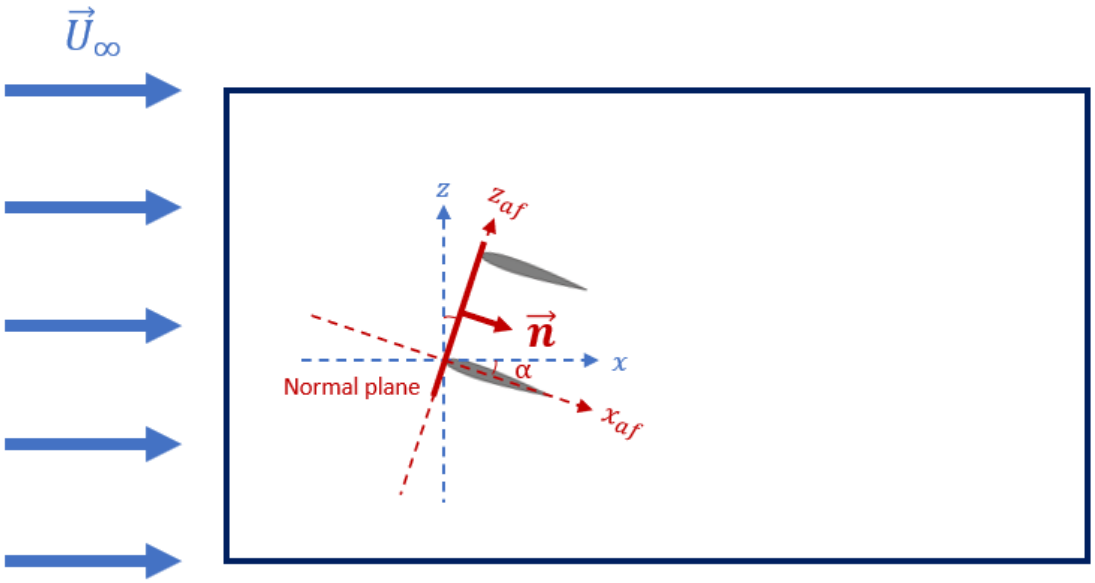


Figure 6.1: Schematic representation of the plane normal to the geometry (Not to scale)

For the C_p and U_x, three different spanwise sections are chosen for analysis. They are named root, the one corresponding to y = 0, mid span, the one corresponding to y = b/2 and tip, the one corresponding to y = b/2-R which is the beginning of the curved region. In terms of C_p, additional graphs on the C_p distribution on the airfoils defined by cutting on the previous regions are presented as well.

For the vorticity ω_n, 8 different planes parallel to the normal plane defined before are examined. This planes are measured from the distance to the trailing edge of the wing and are specified in terms of MAC. The list of planes is: 1.1 MAC, 1.5 MAC, 2 MAC, 3.5 MAC, 5 MAC, 10 MAC, 15 MAC and 20 MAC.

Regarding the resolution of the simulations, the ones corresponding to the high refinement case were used for post-processing.

6.2. POST-PROCESSING RESULTS

Three different points on the lift coefficient curve were chosen for post-processing due to their relevance in aerodynamic analyses. The first one corresponds to the end of the linear region, $\alpha = 12.3\,°$, which is the point where lift performance begins to asymptotically increase as the boundary layer starts to separate. The second one corresponds to the point of maximum lift coefficient, $\alpha = 18.5\,°$, which is the point followed by a sudden decrease in lift due to strong separation of the boundary layer. The final one corresponds to the wing already stalled, $\alpha = 21°$, which is the point where strong non linear effects are expected.

Being separation key in the post process of the results, a brief introduction to airfoil stalling types is presented hereafter. In airfoil aerodynamics, four different types of stall can be distinguished [81][82]. The first one is called thin airfoil stall, affecting airfoils with sharp leading edges. It consists on laminar separation followed by turbulent reattachment which results in a separation bubble that is present at any angle of attack different than zero. This bubble expands with increasing angle of attack forming a benign stall when it reaches the trailing edge. The second type is known as leading edge stall, which is characterized by significant and abrupt lift loss. It is identified by the formation of a short bubble near the leading edge which changes its size with the angle of attack. At critical conditions either the short bubble bursts or turbulent reseparation occurs after the short bubble, causing a sudden loss of lift. The third type of stall occurs when trailing edge stall occurs before leading edge short bubble separation, causing a gradual decrease in lift coefficient curve slope as trailing edge stall progresses. Note that when leading edge stall occurs, an abrupt decrease in lift is expected. The fourth and last type corresponds to pure trailing edge separation, which gradually decreases the lift as the trailing edge separation point moves towards the leading edge of the airfoil. Figure 6.2 provides an illustration of the first three types of stall.

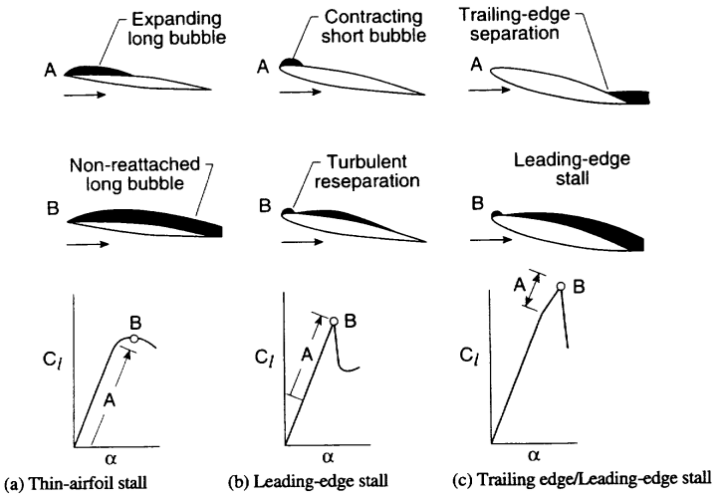


Figure 6.2: Airfoil stall classification by Gault [81]

6.2.1. END OF THE LINEAR REGION

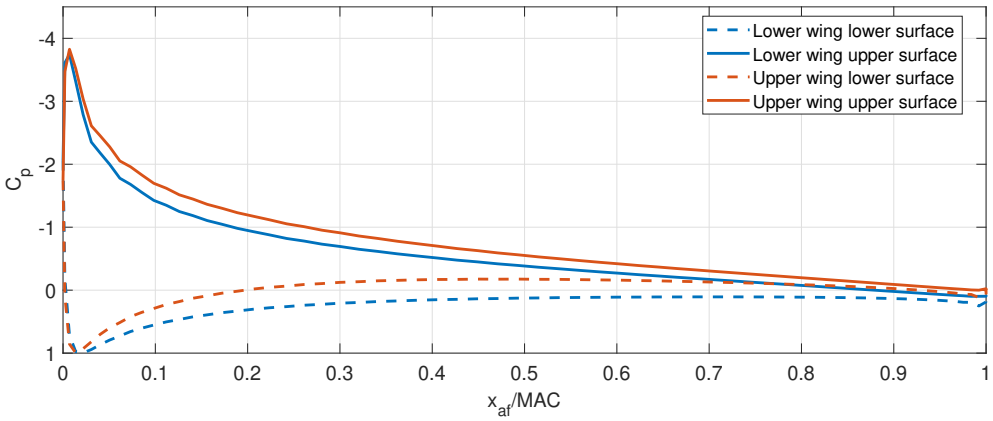
Regarding the pressure coefficients over airfoils of the previously mentioned wing sections, they are depicted in Figure 6.3. Figures 6.3a & 6.3b show conventional airfoil C_p trends in the linear region at the root and mid span sections. This means, high suction peaks near the leading edge and pressure coefficient near 0 at the trailing edge, which are characteristics of attached flow. Regarding the tip section depicted in Figure 6.3c, a smaller suction peak is found than in the previous sections, but also a localized separation and reattachment at the upper wing upper side is observed. Although this separation is not critical, it would indicate the beginning of leading edge stall.

Regarding the C_p and U_x in the different sections, they are depicted in Figure 6.4. Again, root and mid section present similar features, high acceleration regions in their upper surfaces and similar stagnation regions at the leading edge of their lower surfaces (Figures 6.4a & 6.4c). At the trailing edge, pressure tends to equalize freestream pressure ($C_p = 0$) and wake lines exit the trailing edge smoothly. Note as well that lower wing has an influence in the pressure experienced by the lower surface of the upper wing, making it lower than the pressure experienced by the lower surface of the lower wing. In terms of U_x negative values appear in the leading edge stagnation points where some recirculation is expected (6.4b & 6.4d). Tip section present smaller acceleration and stagnation regions, which are depicted in Figure 6.4e. Upper wing wake does not equalize with the freestream pressure in this plane and neither the wake direction keeps constant but deflects upwards (Figure 6.4f). This might be caused by the particular location of this section. Since this is a region followed by a rotation and thickness reduction of the profile (from NACA 0012 to NACA 0003), out of the section plane effects are expected.

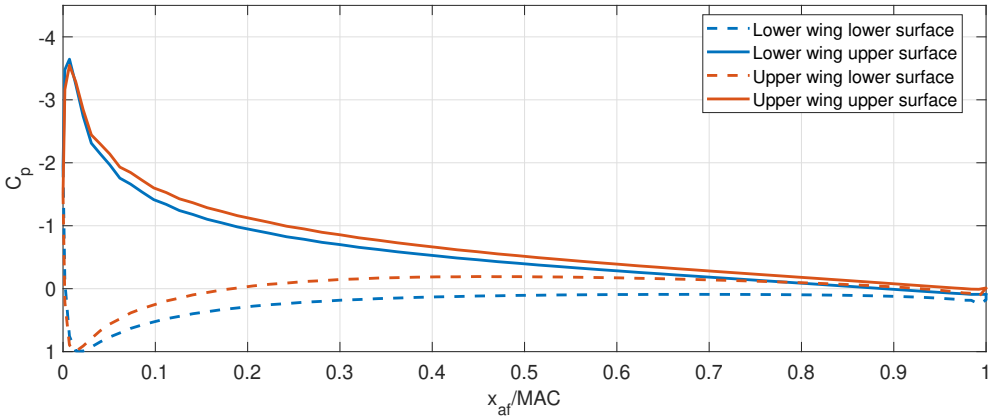
In terms of vorticity, the sequence of normal planes is depicted in Figure 6.5. In general, the trend of vortex generation is similar to that found in conventional wings with winglets: counter clockwise vortices at the right part and clockwise vortices at the left part of the wing. However, since the winglet connects both upper and lower wing, it does not allow the enlargement of tip vortices, but eventually fuses both vortices from upper and lower tips into one. A relevant feature to highlight in this vorticity contours can be found in Figures 6.5b & 6.5c, which is the presence of counter rotating small vortices at the tip section (y = b/2-R) of the upper wing. These are caused due to the concave shape of the upper wing upper surface, where the flow struggles to reattach to the upper surface.

6.2.2. MAXIMUM LIFT COEFFICIENT

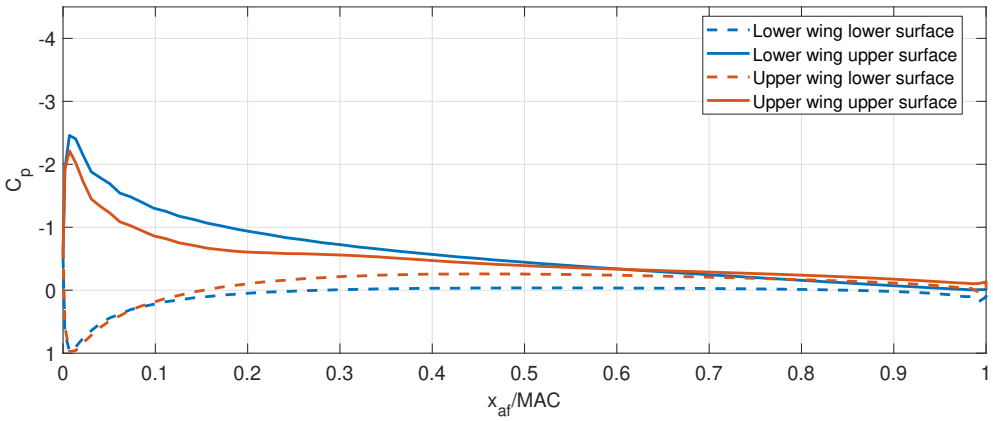
In this case, some regions are experiencing strong separations and some still keep flow attached. On the one hand, pressure coefficient at the root (Figure 6.6a) shows boundary layer separation at the upper wing leading edge in the form of a huge separation bubble extending up to the trailing edge. On the other hand, massive separation occurs at the lower wing root section making pressure at the trailing edge not matching the one at the freestream. Pressure coefficient at the midspan (Figure 6.6b) presents two opposite trends: an upper wing with fully attached flow and a lower wing with boundary layer separation. Finally, the tip section (Figure 6.6c)



(a) Root



(b) Mid span

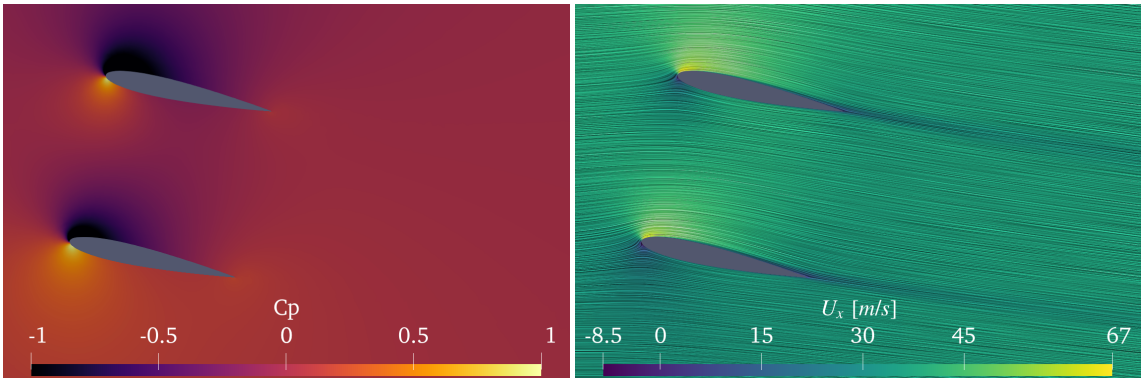


(c) Tip

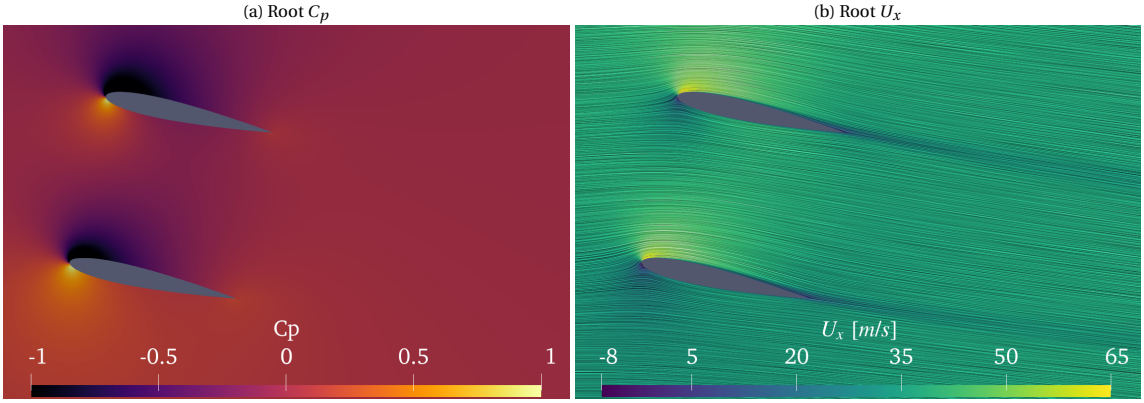
Figure 6.3: Airfoil C_p distribution at different sections for α = 12.3°

presents some flow separation on the second half of the lower wing and an increasing leading edge separation bubble on the upper wing.

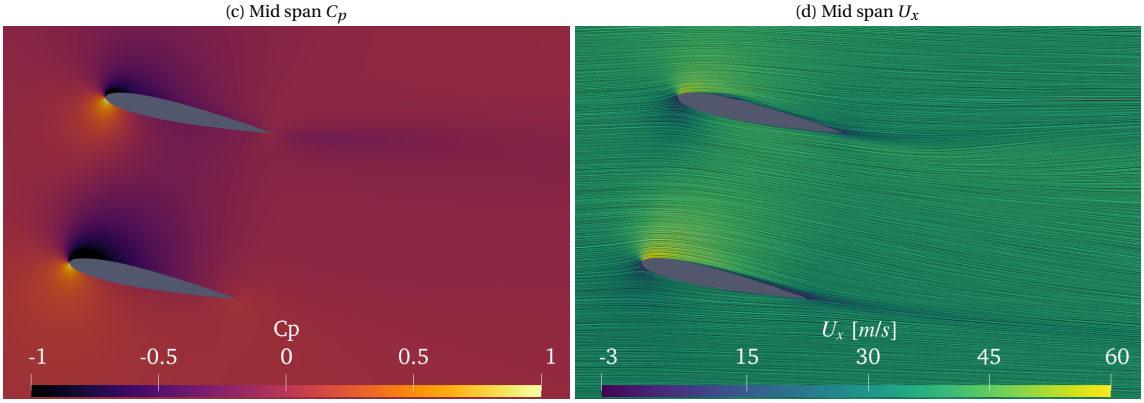
The pressure coefficient and the velocity component aligned with the freestream are depicted in Figure 6.7. Figures 6.7a & 6.7b show that the flow is experiencing separation on both wings, being the lower one under the most critical condition (massive separation) and the upper one alleviated by the lower one (lower pressure on upper wing lower surface) showing a huge separation bubble. Figures 6.7c & 6.7d show a less critical condition where separation is only occurring on the lower wing in the form of a separation bubble.



(a) Root C_p (b) Root U_x



(c) Mid span C_p (d) Mid span U_x

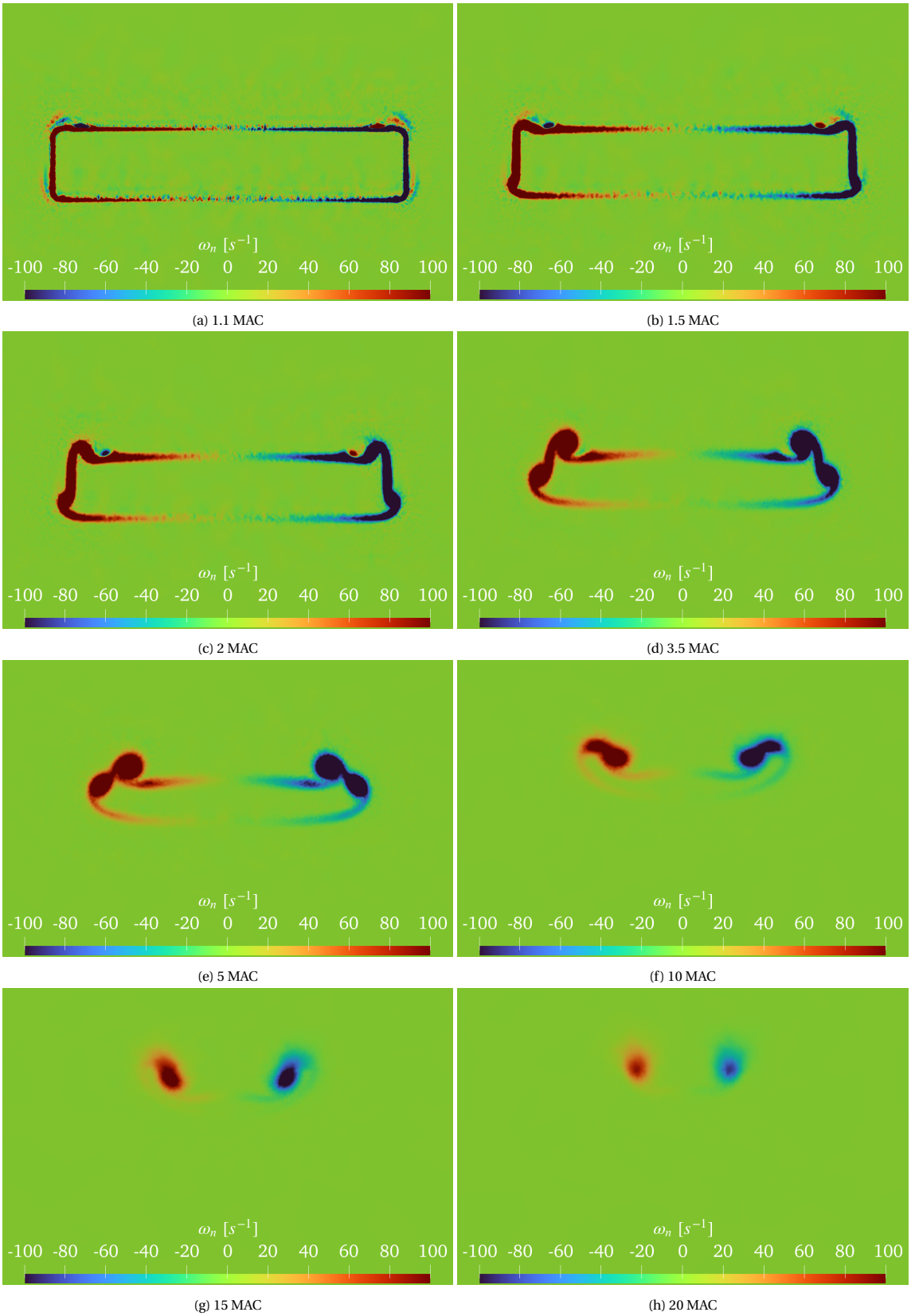


(e) Tip C_p (f) Tip U_x

Figure 6.4: C_p and U_x contours at different span sections for $\alpha = 12.3°$

Both trailing edge wakes point in the direction that allows pressure equalizing with the freestream one. Finally, Figures 6.7e & 6.7f show a some degree of boundary layer separation of the lower wing starting at the trailing edge and the further development of leading edge stall on the upper wing. As in the previous flight condition, pressure is not matched for the upper wing and the wake deflects upwards.

In terms of vorticity, having some degree of separation makes the flow turbulent and prone to generate vortices inside and outside these regions. Figure 6.8 depicts the sequence of vorticity in the normal planes. The final result is again the one that could be expected in a conventional wing after clockwise and counter clockwise vortices fuse together. However Figure 6.8a present a wide variety of vortices. The separated regions tend to generate vortices opposite sense than the ones generated by the wings themselves. Outside these separation regions, opposite vortices to the ones generated inside are created to keep constant circulation. Note that although the size of these vortices is proportional to the separated regions, their strength is lower than the natural vortices released by the wing structure itself. This is why in the recombination the prevailing sense of rotation is kept as the one from the vortices generated by the wing.



(a) 1.1 MAC

(b) 1.5 MAC

(c) 2 MAC

(d) 3.5 MAC

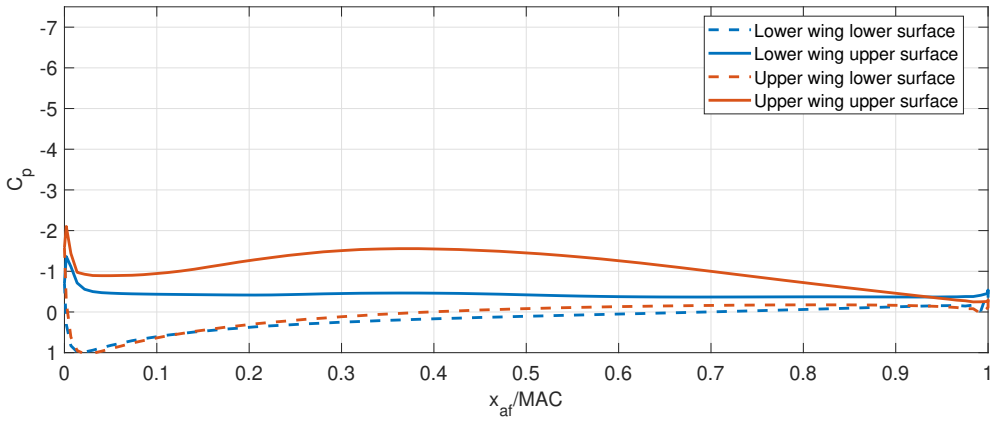
(e) 5 MAC

(f) 10 MAC

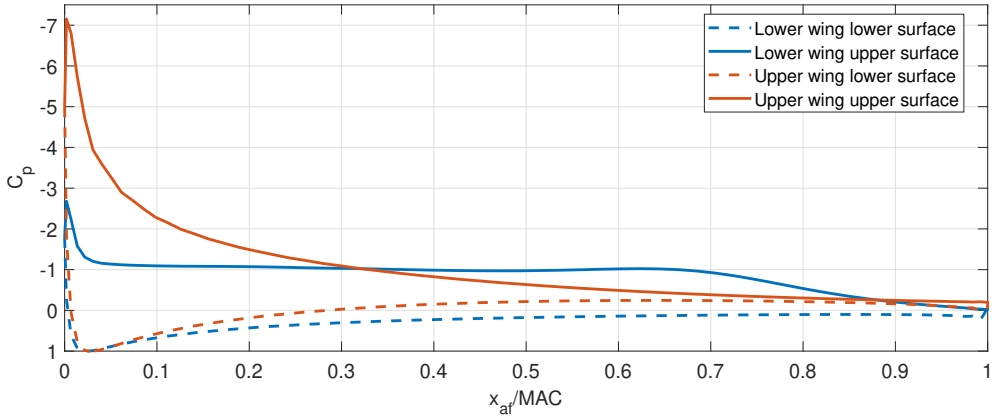
(g) 15 MAC

(h) 20 MAC

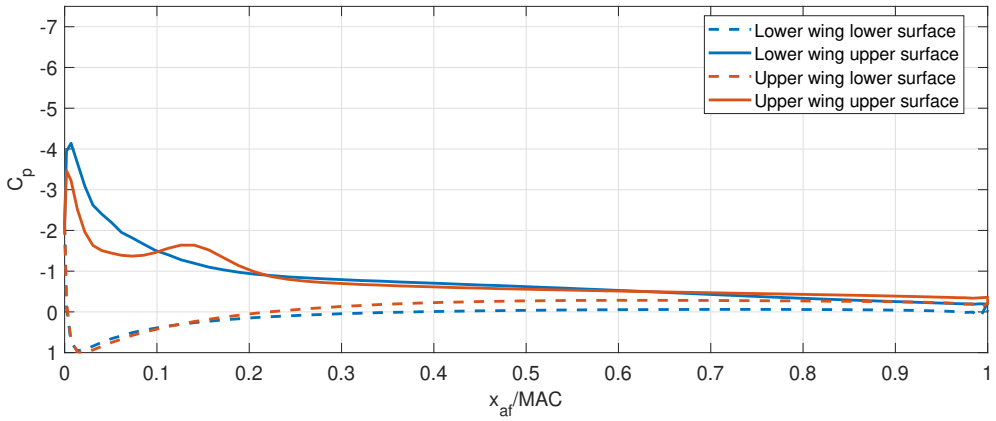
Figure 6.5: Vorticity contours for α = 12.3°



(a) Root



(b) Mid span

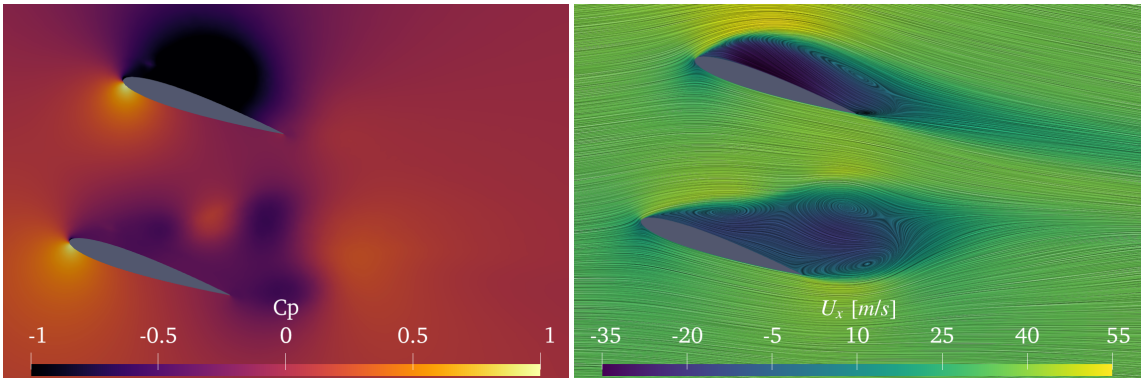


(c) Tip

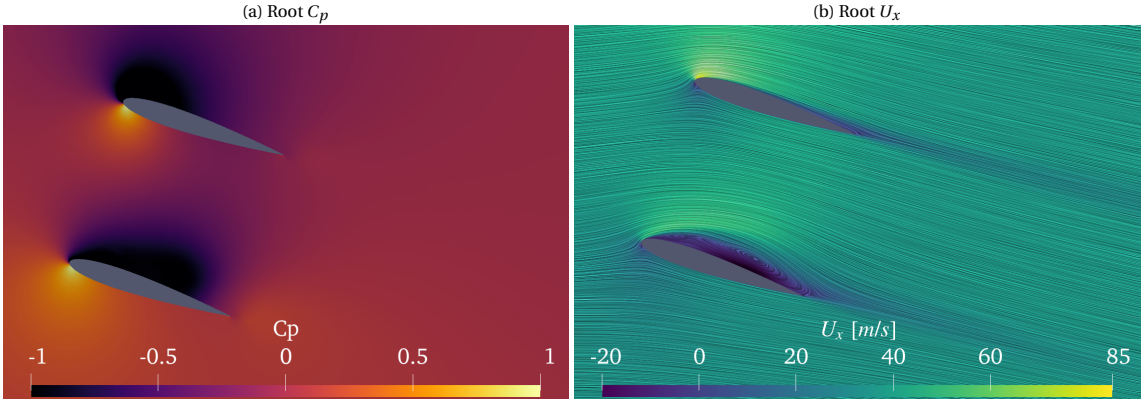
Figure 6.6: Airfoil C_p distribution at different sections for α = 18.5°

6.2.3. STALLED WING

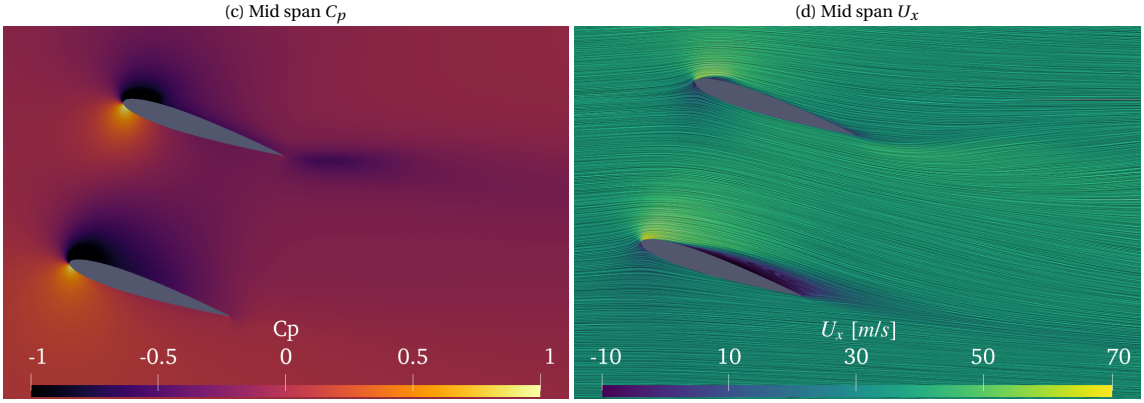
In this case, most of the wing is stalled and lift coefficient has decreased dramatically. At the wing root and mid section, massive stall is experienced by both wings, being the pressure coefficient almost constant for the upper surfaces after the suction peak (Figures 6.9a & 6.9b). However, for the case of the wing tip section, the leading edge stall separation bubble in the upper wing has not turned into massive separation yet, meaning that it is still providing some useful lift force (Figure 6.9c). For the lower wing separation has developed in a less abrupt way than for the other two sections.



(a) Root C_p (b) Root U_x



(c) Mid span C_p (d) Mid span U_x

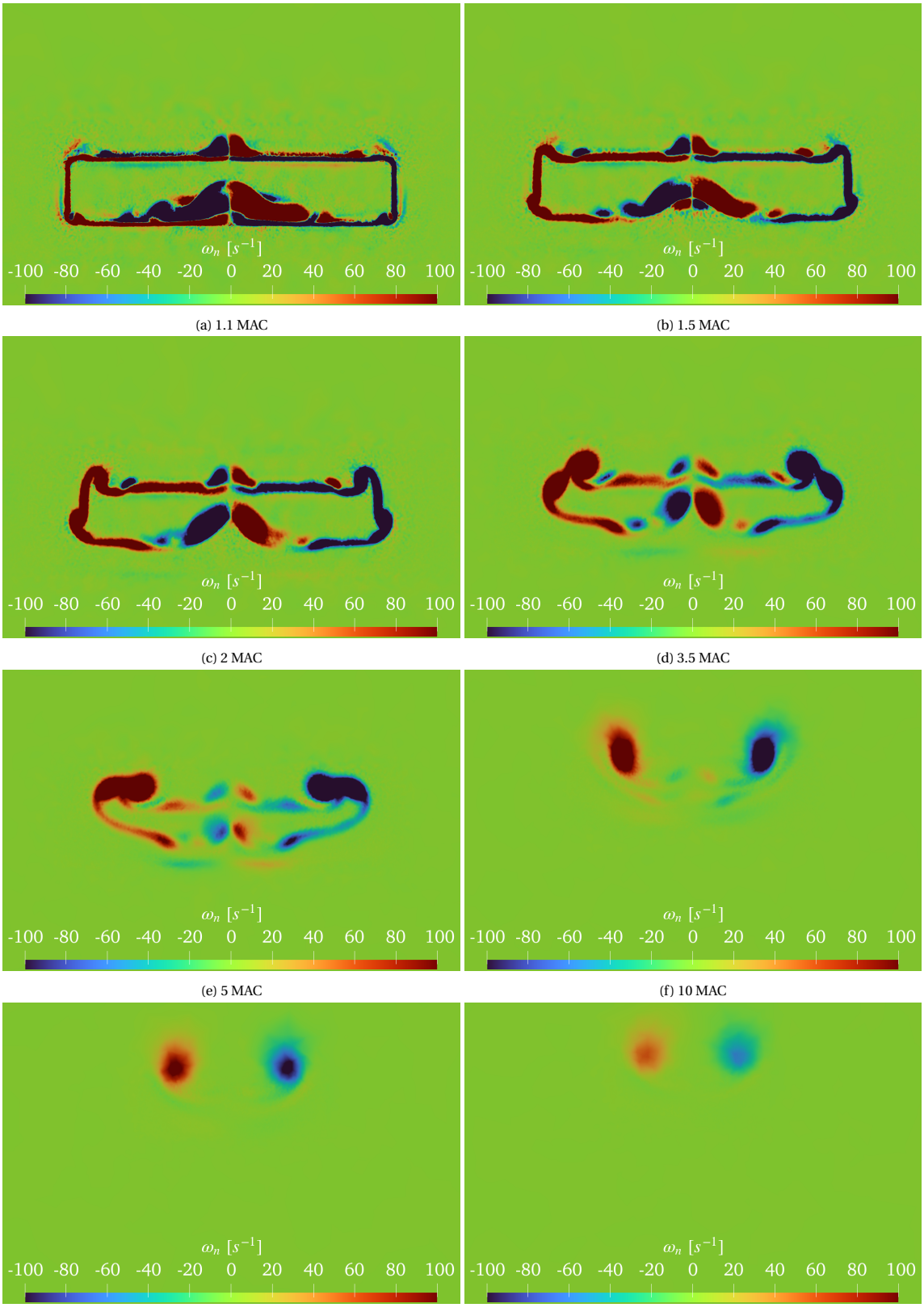


(e) Tip C_p (f) Tip U_x

Figure 6.7: C_p and U_x contours at different span sections for α = 18.5°

Figure 6.10 depicts the pressure coefficient and the velocity component aligned with the freestream. At the root, Figures 6.10a & 6.10b show large recirculation regions in the trailing edge of the upper wing given by the massive boundary layer separation. The lower wing presents smaller recirculation regions and pressure match is not achieved in either wing. At the mid span section, 6.10c & 6.10d show similar separation regions that try to align with the chordwise airfoil direction. At the tip, Figures 6.10e & 6.10f show the difference between the stalling trends of the upper and lower wings at this section. As previously stated just the upper wing is the one not completely stalled, whereas the size of the separated region of the lower wing seems to be lower as compared to the size of the other two sections. This lower wing has fully developed trailing edge stall.

In terms of vorticity, Figure 6.11 shows the evolution of vortices in the normal planes. In this flight condition, boundary layer separation occurs in the whole lower wing and in most of the upper wing but the region nearby the upper wing tips. Huge regions of non-zero vorticity appear in the inner and outer part of the separated boundary layer changing rotation sense to preserve the total circulation (Figure 6.11a). The most relevant fact is the final rotation sense of the vortices is opposed to the one found in the previous two flight



(a) 1.1 MAC

(b) 1.5 MAC

(c) 2 MAC

(d) 3.5 MAC

(e) 5 MAC

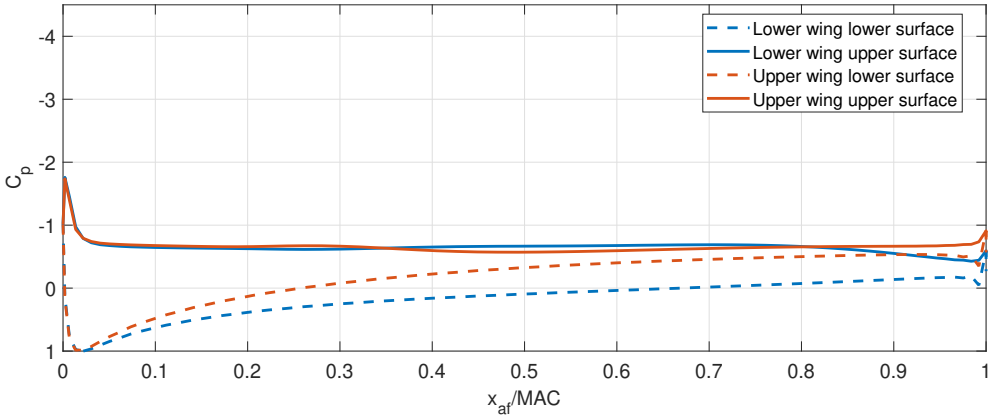
(f) 10 MAC

(g) 15 MAC

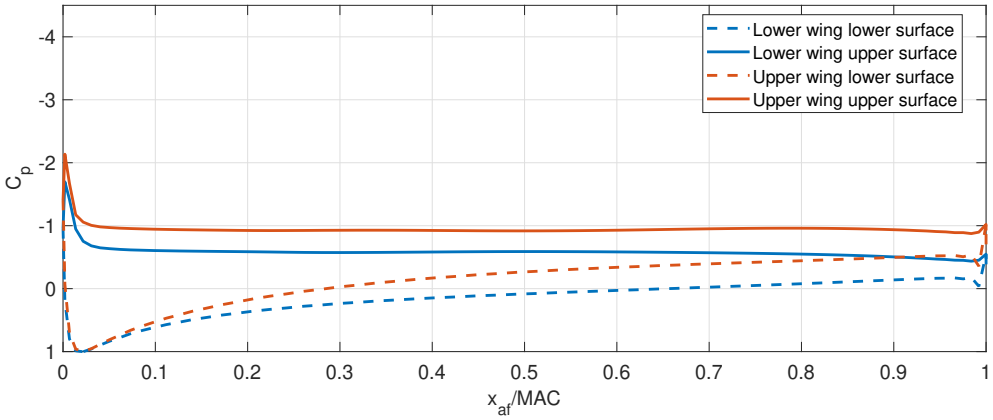
(h) 20 MAC

Figure 6.8: Vorticity contours for α = 18.5°

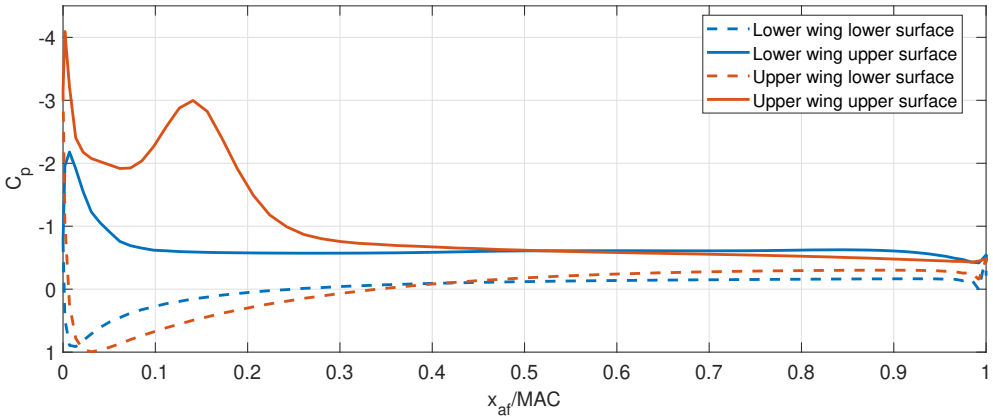
conditions. Since the wing is operating in a highly non linear and stalled regime, turbulence, separation and angle of attack are leading to this non conventional results.



(a) Root



(b) Mid span

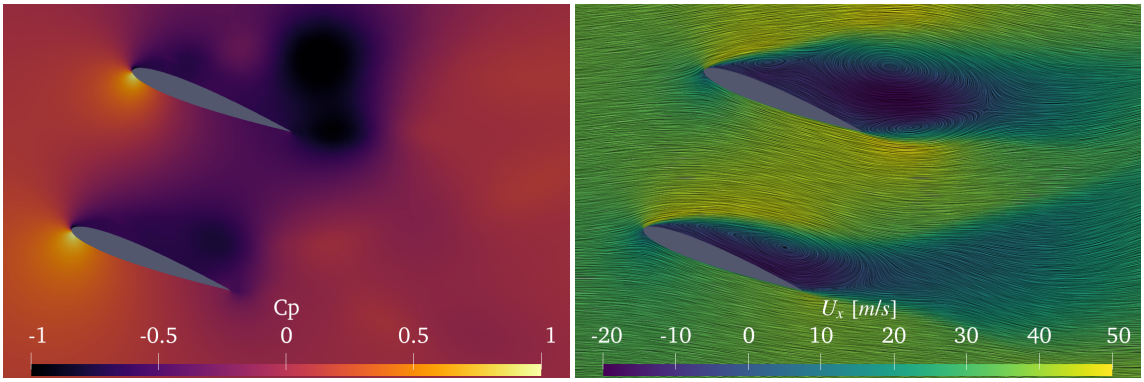


(c) Tip

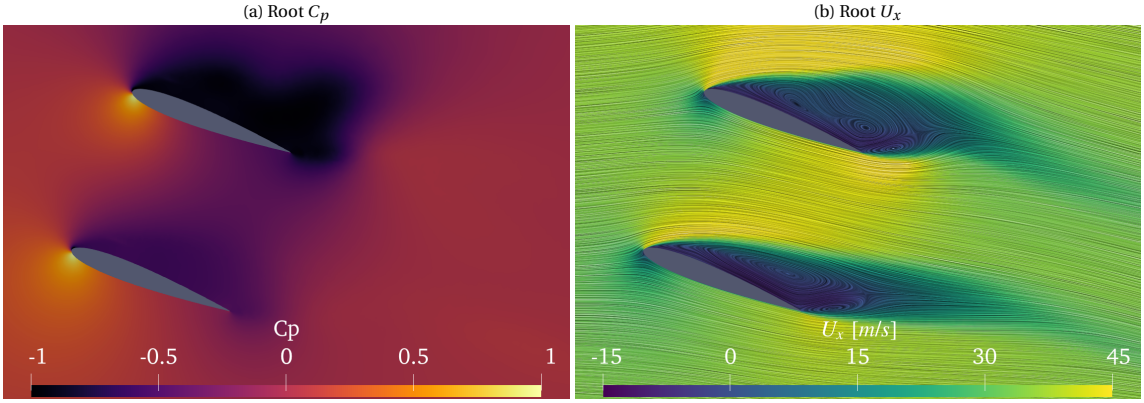
Figure 6.9: Airfoil C_p distribution at different sections for $\alpha = 21°$

6.3. RESULTS DISCUSSION

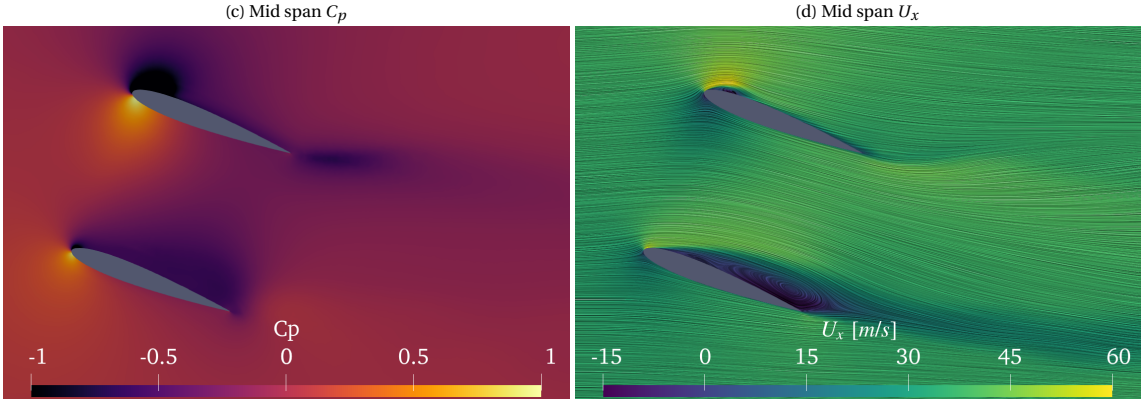
After having analyzed the results from the simulations, some conclusions can be extracted about the aerodynamics of box wing designs with no stagger and symmetric airfoils. The first of them is that the root section



(a) Root C_p (b) Root U_x



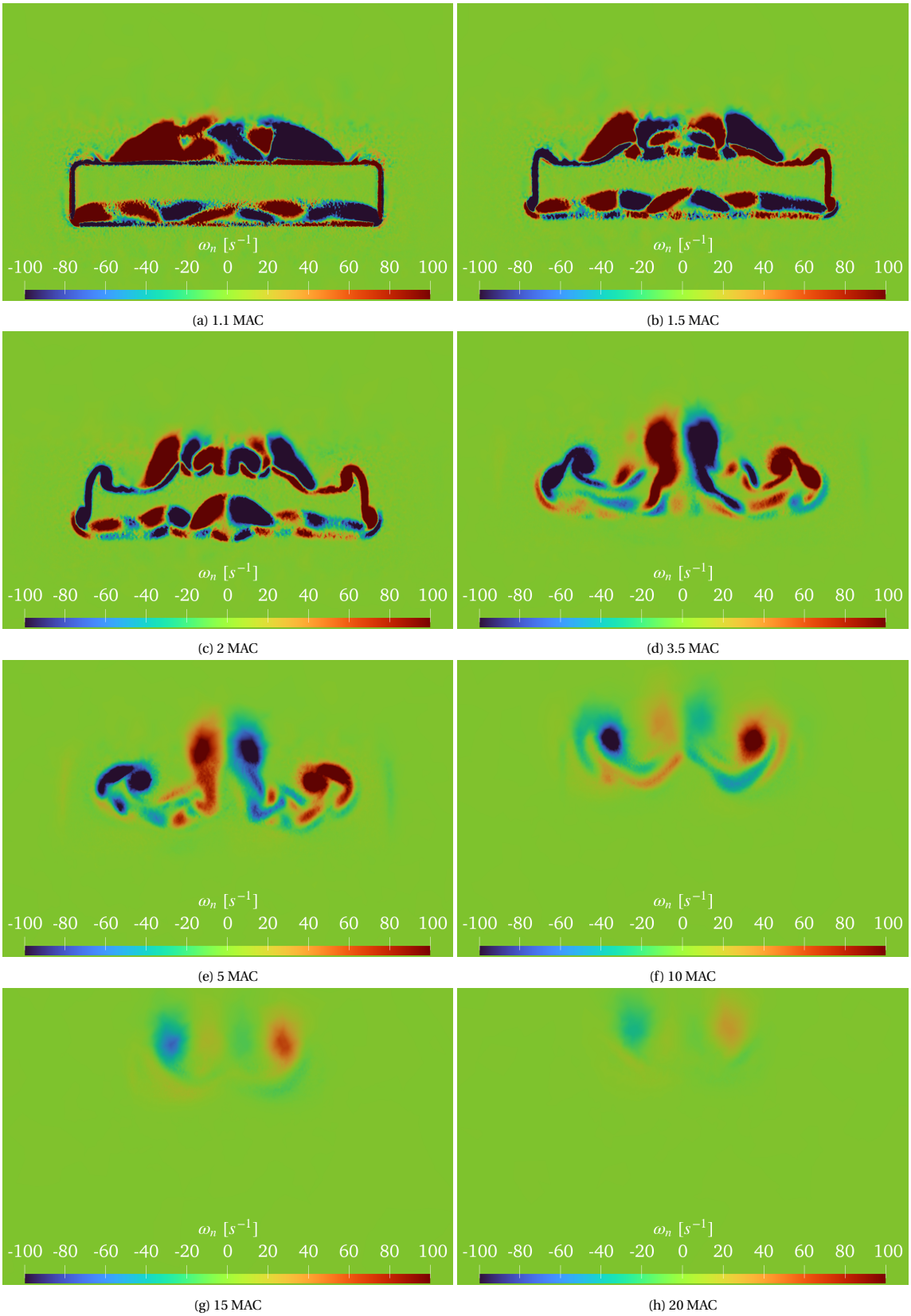
(c) Mid span C_p (d) Mid span U_x



(e) Tip C_p (f) Tip U_x

Figure 6.10: C_p and U_x contours at different span sections for $\alpha = 21°$

stalls before the tip section. The main difference between these sections is the cant radius region near the tip inducing 3D effects. In this case these effects are beneficial and prevent the whole wing entering in stall at once. The second conclusion is about the relative size of the separation between upper and lower wing. As a general trend, upper wing stall is delayed with respect to lower wing stall, this might be explained because of the reduced pressure acting at the lower surface of the upper wing and the effective angle of attack of the flow, which may be changed in the region between wings. The third conclusion is that regardless of the vortices generated by the separation region, the direction that prevails in the end is the one imposed by the wing perturbating the flow. However, separation generated vortices can influence on these rotation sense. Finally, the stall mechanism that governs the whole wing is thin airfoil stall, except on the regions nearby the upper wing tip where leading edge stall and trailing edge stall are occurring.



(a) 1.1 MAC

(b) 1.5 MAC

(c) 2 MAC

(d) 3.5 MAC

(e) 5 MAC

(f) 10 MAC

(g) 15 MAC

(h) 20 MAC

Figure 6.11: Vorticity contours for $\alpha = 21°$

7

PARAMETRIC STUDY AND OPTIMIZATION

This chapter aims to describe the process for achieving an optimized box wing design equivalent to a conventional wing design for AWE systems. The chapter begins with the definition of equivalent wings, explaining the features that should be maintained and the reference conventional wing chosen. The chapter continues with a parametric study to check for the aerodynamic variations in terms of lift coefficient that result from sweeping on different geometrical parameters. This chapter concludes with a detailed explanation of the optimization process and a discussion of the extent of validity of the final results.

7.1. EQUIVALENT WING

In order to obtain a box wing design which is equivalent to a conventional wing design for AWE applications, the criterion for equivalence must be established beforehand. In this particular work, an equivalent wing must fulfill five conditions:

1. Provide an equal or higher tether force at all angles of attack, which is equivalent to the lift coefficient curve C_L vs α.
2. Use the same airfoil profiles in the lifting regions (upper and lower wings).
3. Wingspan equal or lower than the conventional wing.
4. Reference surface equal or lower than the conventional wing.
5. Weight equal or lower than the conventional wing.

Notice that these conditions give degrees of freedom to the design process, with a wide range of possible variables of interest to be swept over.

The conventional wing geometry chosen was defined by Eijkelhof [44], who presented a multi megawatt AWE reference system within his optimization framework. The relevant geometrical parameters regarding the wing of this system and the flight conditions are depicted in Tables 7.1 & 7.2. Geometrical parameters from this design are referred with the subscript CW onwards. Note that the airfoil is an uncommon design that was obtained through reverse engineering and improvement of the Makani M600 wing sections. Figure 7.1a illustrates this high lift highly cambered airfoil. Figure 7.1b shows the C_L vs α curve of the conventional wing design obtained with APAME. Altough this curve is the result of several Fluid Structure Interaction (FSI) iterations meaning that it corresponds to the deformed wing shape and not the original one, it is assumed that box wing rigidity is superior or at least it can be improved with more connections between wings with little deformation. This is the reason why it is be directly compared to the deformed conventional wing lift curve. This curve is defined by its slope $C_{L\alpha}$ = 6.0798 and α = 0 lift coefficient value C_{L0} = 0.9076 ranging from α = -14.8° to α = 9.2°.

7.2. PARAMETRIC STUDY

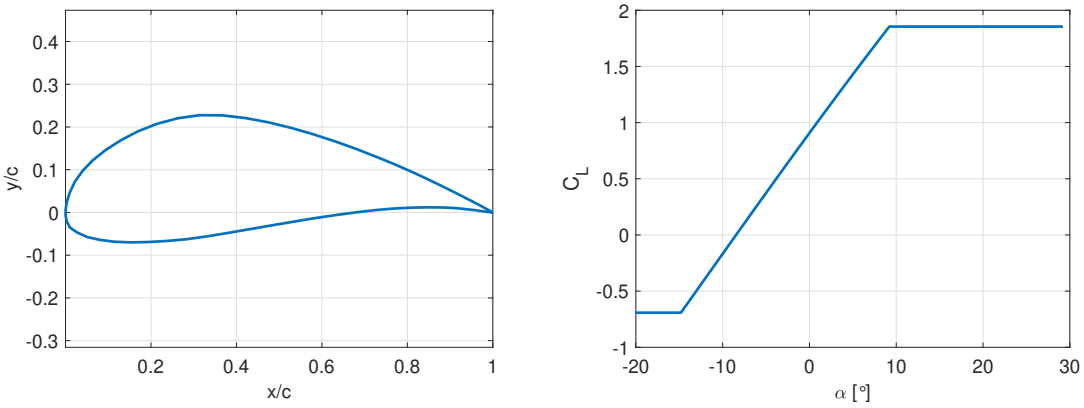
In order to decide which parameters is the lift coefficient more sensitive to, some sweeps on wingspan, height over span ratio, stagger over chord ratio, fore wing area and taper ratio were performed fixing the other parameters.

b [m]	42.4694
MAC [m]	3.5425
S_{ref} [m^2]	150.4478
Area skin [m^2]	323.3616
Airfoil name	MRevE-v2$_{HC}$
Airfoil (t/c) @ (x/c)	28.6% @ 31.4%
Airfoil (f/c) @ (x/c)	9.0% @ 41.1%

Table 7.1: Geometrical parameters for the conventional wing [44]

U_∞ [m/s]	60
T_∞ [K]	288.15
ρ_∞ [kg/m^3]	1.228
ν [kg/ms]	1.4603E-5

Table 7.2: Reference conditions for the conventional wing [44]



(a) MRevE-v2$_{HC}$ airfoil

(b) 3D Lift coefficient curve

Figure 7.1: Conventional wing design features from Eijkelhof [44]

The metric that was used to decide the sensitivity to C_L is computed as the area difference between the two lift curves: box wing (BW) and conventional wing (CW), following Eq. 7.1, where α_i= -14.8° and α_f= 9.2°.

$$I = \int_{\alpha_i}^{\alpha_f} \left(C_{L,BW}(\alpha) - C_{L,CW}(\alpha) \right) d\alpha \tag{7.1}$$

Considering the 3D panel method predicts the lift coefficient in the linear region, Eq. 7.1 can be further simplied to Eq. 7.2. Note that this equation is valid whenever the intersection angle of attack between the curves α_I lies within the integration limits.

$$I = \frac{1}{2}(\alpha_I - \alpha_i)\left(C_{L,BW}(\alpha_i) - C_{L,CW}(\alpha_i)\right) + \frac{1}{2}\left(\alpha_f - \alpha_I\right)\left(C_{L,BW}\left(\alpha_f\right) - C_{L,CW}\left(\alpha_f\right)\right) \tag{7.2}$$

For the case where α_I lies out of the integration bounds, Eq. 7.1 adopts the shape of Eq. 7.3.

$$I = \frac{1}{2}(\alpha_F - \alpha_i)\left(\left(C_{L,BW}(\alpha_i) - C_{L,CW}(\alpha_i)\right) + \left(C_{L,BW}\left(\alpha_f\right) - C_{L,CW}\left(\alpha_f\right)\right)\right) \tag{7.3}$$

Notice that the sign of I, determines whether the box wing design lift coefficient curve has a better overall performance than the conventional wing design (+) or not (-).

First sweep was performed changing the wingspan and calculating the chord of the box wing. For this calculation, the reference surface of the box wing was fixed to the same value the conventional wing had. As an additional simplification, the chord was kept constant at all box wing sections, thus being equivalent to the MAC. The other variables were kept fixed during this sweep. Eq. 7.4 was used to obtain the MAC. Values

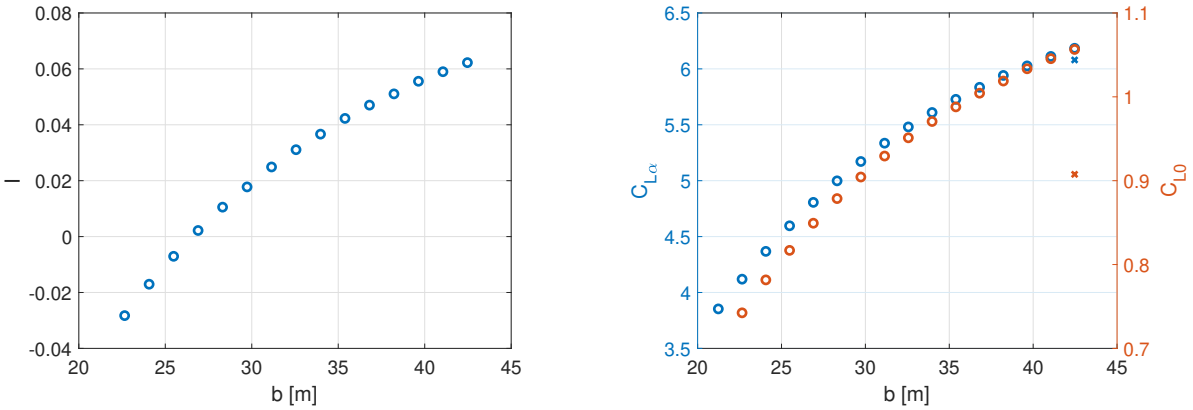
for this sweep are shown in Table 7.3.

$$MAC = \frac{S_{ref,CW}}{2b} \tag{7.4}$$

b [m]	Start: $b_{CW}/2$ End: b_{CW} Points: 16
MAC [m]	Eq. 7.4
h/b	0.2
s/MAC	0.3
Airfoilwings	MRevE-v2$_{HC}$
Airfoil winglet	NACA 0030
R [%]	15

Table 7.3: Geometrical parameters sweeping on wingspan

Figure 7.2a depicts the evolution of the lift coefficient difference with the wingspan. Two conclusions can be extracted from this graph. Being both the height to span and stagger to chord ratios fixed, an increase in wingspan implies an increase in the overal lift coeffficient of the wing. The second one is that being the reference area fixed, the chord will tend to decrease with increasing wingspan, and the associated increase in lift coefficient will tend to increase asymptotically. Both $C_{L\alpha}$ and C_{L0} also increase in the same manner. This is depicted in Figure 7.2b, where crosses denote the values from the conventional wing design. On the one hand, for this particular conditions the value of $C_{L0,CW}$ is exceeded whenever b_{BW}>0.7b_{CW}, which would be equivalent to an increase in camber of the conventional wing sections. On the other hand, the slope $C_{L\alpha}$ follows the trend of increasing whenever the aspect ratio of the box wing increases and reach similar values when both aspect ratios are the same.



(a) I sensitivity

(b) $C_{L\alpha}$ and C_{L0} sensitivity

Figure 7.2: Sweep on wingspan

The second sweep was performed varying independently two parameters: the stagger to chord ratio s/c and the height to span ratio h/b. Again the reference surface of the box wing is kept fixed to the same value as the conventional wing. Table 7.4 includes the different sweeps performed in this case. Figure 7.3a depicts the sensitivity of I to the previously mentioned sweeps. It can be observed that at least two optimum regions in terms of I exist: the first one with slightly negative stagger to chord ratios around -0.5 and ratios of h/b around 0.6; and the second one with similar stagger but low h/b ratio around 0.2. However, looking at the C_{L_α} and C_{L0} contours in Figures 7.3b & 7.3c it is clear that any of the lift coefficient curves obtained from the sweeps is exceeding the one from the conventional wing at all angles of attack. This occurs because the intersecting angle of attack α_I lies within the range [-14.8,9.2], and because the aspect ratio of the wing is fixed, leading to little improvement in $C_{L\alpha}$. However, the tendency towards having two independent wings (high h/b) shows some improvement in terms of $C_{L\alpha}$. Different is the case for C_{L0}, which shows a optimal region centered in

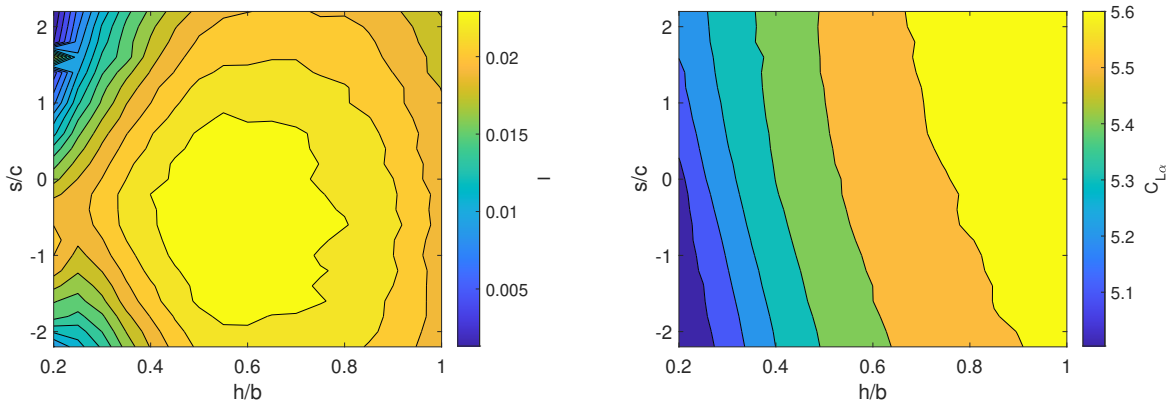
sligthly negative s/MAC around -0.5 and h/b around 0.75. This can be explained due to the influence between the two wings.

There are certain combinations (yellow regions) where the influence of the two wings is equivalent to have an effective camber that is better than the two independent cambers on each of the separate wings. This effect leads to higher values of C_{L0}. For values outside of this region, either the effective camber is non optimal, or it becomes equivalent to the actual camber of the corresponding wings (wings act independently).

Notice that in this second sweep, h/b values higher than 1 were not explored because this configuration was not considered to be a practical solution.

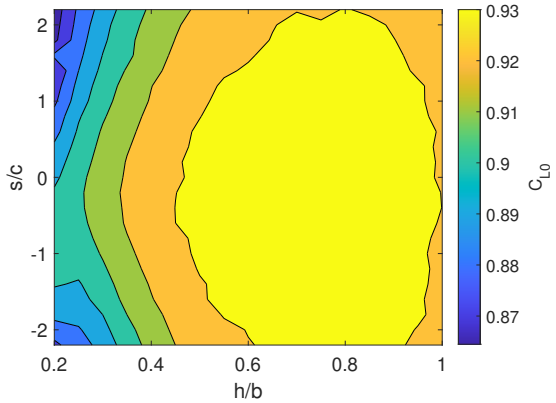
b [**m**]	$0.7b_{CW}$
MAC [**m**]	2.5304
h/b	Start: 0.2 End: 1 Points: 17
s/MAC	Start: -2.2 End: 2.2 Points: 41
Airfoilwings	MRevE-v2$_{HC}$
Airfoil winglet	NACA 0030
R [%]	15

Table 7.4: Geometrical parameters sweeping on s/c and h/b



(a) I contour

(b) $C_{L\alpha}$ contour



(c) C_{L0} contour

Figure 7.3: Sweeps on h/b and s/c

The third and last sweep was performed varying independently two parameters: the taper ratio λ and

the fore wing area percentage S_l/S_{ref}, keeping the reference surface constant and equal to the conventional wing. For simplicity. the taper ratio variation was the same for both upper and lower wings and it was defined as Eq. 7.5. c_{ti} and c_{ri} denote chord at the root and the tip of the corresponding wing (upper or lower).

$$\lambda_i = \frac{c_{ti}}{c_{ri}} \tag{7.5}$$

Since both parameters in this last swept are strongly related to the reference surface, some operations need to be defined to compute the chords at the different sections of the box wing. Considering upper and lower wings planforms have trapezoidal shape, Eqs. 7.6, 7.7, 7.8 & 7.9 define the relevant chords.

$$c_{r1} = 2\frac{S_l}{S_{ref,CW}}\frac{S_{ref,CW}}{b(1+\lambda_1)} \tag{7.6}$$

$$c_{t1} = \lambda_1 c_{r1} \tag{7.7}$$

$$c_{r2} = 2\left(1-\frac{S_l}{S_{ref,CW}}\right)\frac{S_{ref,CW}}{b(1+\lambda_2)} \tag{7.8}$$

$$c_{t2} = \lambda_2 c_{r2} \tag{7.9}$$

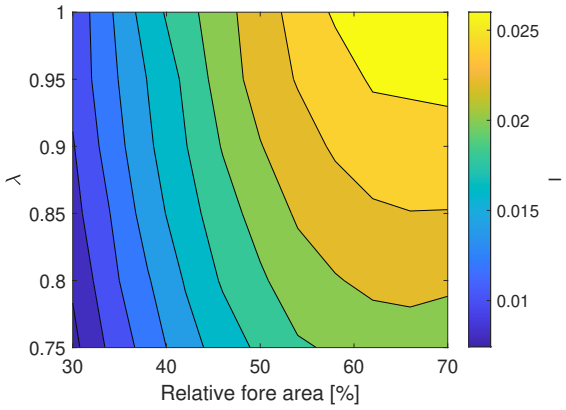
Table 7.5 contains the geometrical parameters for this last sweep. Figure 7.4a shows the trend of the integral difference between curves (I) for this last sweep. In particular, it depicts that the highest positive difference is achieved for rectangular planform wings (λ = 1) and high fore wing area percentage. However, this trend is mainly dominated by the value of C_{L0} as depicted in Figure 7.4c. If the maximum lift coefficient slope was desired, the best option would be to keep rectangular wings with the same surfaces. Figure 7.4b presents the highest values of $C_{L\alpha}$ being reached when the relative fore area is between 50-55%.

b [**m**]	$0.7b_{CW}$
$\mathbf{c_{r1}}$ [**m**]	Eq. 7.6
$\mathbf{c_{t1}}$ [**m**]	Eq. 7.7
$\mathbf{c_{r2}}$ [**m**]	Eq. 7.8
$\mathbf{c_{t2}}$ [**m**]	Eq. 7.9
h/b	0.75
s/MAC	1.6
λ	Start: 0.75 End: 1 Points: 6
$\mathbf{S_l/S_{ref,CW}}$	Start: 0.3 End: 0.7 Points: 11
Airfoilwings	MRevE-v2_{HC}
Airfoil winglet	NACA 0030
R [%]	15

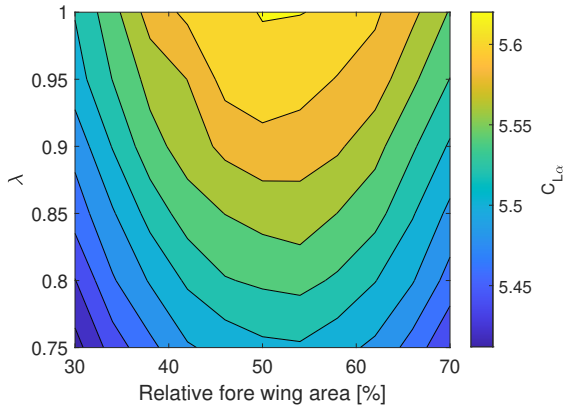
Table 7.5: Geometrical parameters sweeping on λ and $S_l/S_{ref,CW}$

Some conclusions in terms of relevant parameters are extracted from the three different sweeps made. The first one is that regardless of the ratios they might be involved in, the wingspan and the chord of the box wing are key parameters for the generation of lift. The second one is that in aerodynamic terms, a high height over span ratio is desired to increase the lift coefficient slope, whereas negative stagger to chord ratio values between 0 and -1 provide the best C_{L0} for a given height to span ratio. Finally, the taper ratio and relative fore wing area were found to be non relevant for this study, so rectangular wings with the same wing planform are chosen for the following studies.

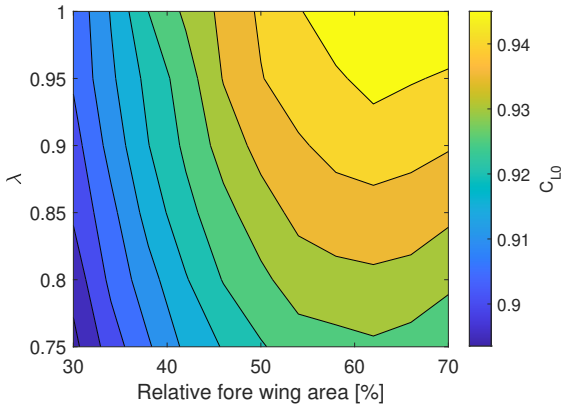
Notice that this parametric study did not take into account structural considerations but aerodynamic ones and was performed to check for the relevant parameters and main trends of the lift coefficient.



(a) I contour



(b) $C_{L\alpha}$ contour



(c) C_{L0} contour

Figure 7.4: Sweeps on λ and $S_l/S_{ref,CW}$

7.3. OPTIMIZATION

Having analyzed the sensitivity of the lift coefficient on some geometrical parameters, a basic optimization study is made to obtain the best lift coefficient performance box wing with the minimum weight. Since it is not the purpose of this work to create an structural model of the box wing, the weight is computed making some simplifications. First, no ribs are considered, and only the skin weight is taken into account. Second, the thickness of the skin, t, is assumed constant throughout the whole wing. Third and last, the material and thus the density of the skin ρ_s is assumed to be the same for the whole wing. Eq. 7.10 gives the simplified weight of the wing, being A the area of the wing skin.

$$W = tA\rho_s \tag{7.10}$$

Notice that since both conventional and box wing are assumed to be made of the same material and have the same skin thickness, the weight comparison reduces to compare their respective wing skin areas.

The steps followed for the optimization process are described as follows:

1. Initialize sweeps on b, c, h and s.
2. Save the combinations that fulfill the constraints on the reference surface and wing weight: A) $S_{ref,BW} \leq S_{ref,CW}$ & B) $A_{BW} \leq A_{CW}$
3. Sweep on the previous combinations simulating in APAME.
4. Filter the results from: A) Numerical errors & B) Values such that:

   $C_{L,BW}(\alpha = -14.8°) < C_{L,CW}(\alpha = -14.8°)$ and $C_{L,BW}(\alpha = 9.2°) < C_{L,CW}(\alpha = 9.2°)$

5. Plot the Paretto front showing the values of I and A_{BW}. The inverse of I is represented instead so both A_{BW} and I^{-1} are tried to be minimized.

6. Choose the optimal point (combination) based on engineering judgement.

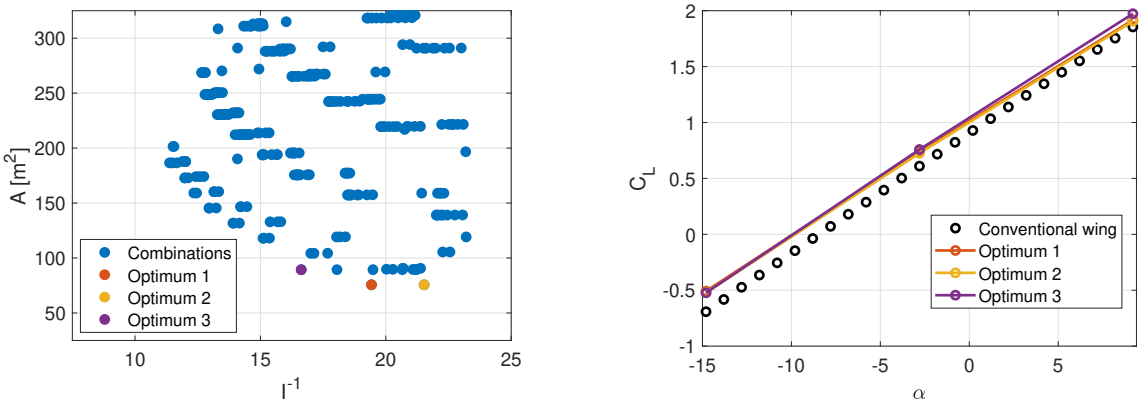
7. Define new sweep intervals (refined) around this combination and repeat from step 2 until step 6.

First initialization of parameters was done following Table 7.6. This gives a total of 10000 combinations that need to be checked for constraint compliance. After this check, 3690 combinations result compliant with the constraints. These combinations are simulated with APAME and subsequently filtered before ploting the Paretto front. This second filtering process ensured 1207 combinations compliant with the constraints and exceeding the conventional wing performance in terms of C_L.

Figure 7.5a depicts the trends in terms of skin area and the inverse of the integral lift coefficient difference. It would be the general practice to choose an optimum satisfying the lowest values of A and I^{-1}. However, since some assumptions were set for the weight being equivalent to the wing skin area and all the combinations depicted exceed the C_L of the conventional wing, optimum values were chosen only regarding the lowest skin area values. Figure 7.5b shows the lift coefficient curves of the conventional wing and the three first optima. The three optimum values in terms of input parameters and results are summarized in Table 7.7.

b [**m**]	Start: $b_{CW}/4$ End: b_{CW} Points: 10
MAC [**m**]	Start: $MAC_{CW}/4$ End: MAC_{CW} Points: 10
h [m]	Start: $0.05b_{CW}$ End: b_{CW} Points: 10
s [**m**]	Start: 0 End: MAC_{CW} Points: 10
Airfoilwings	MRevE-v2$_{HC}$
Airfoil winglet	NACA 0030
R [%]	15

Table 7.6: First sweep on parameters for optimal box wing design



(a) Paretto front

(b) C_L vs α

Figure 7.5: First sweep for optimum values

Second initialization of parameters was done focusing on Optimum 1 and refining around it as specified in Table 7.8. Note that in the previous sweep, negative stagger was omitted to save computational time. Even

Parameters	Optimum 1	Optimum 2	Optimum 3
b [m]	17.6956 ($0.42b_{CW}$)	17.6956 ($0.42b_{CW}$)	21.2347 ($0.5b_{CW}$)
MAC [m]	0.8856 ($0.25MAC_{CW}$)	0.8856 ($0.25MAC_{CW}$)	0.8856 ($0.25MAC_{CW}$)
h [m]	2.1235 ($0.05b_{CW}$)	2.1235 ($0.05b_{CW}$)	2.1235 ($0.05b_{CW}$)
s [m]	0	0.3936 ($0.11MAC_{CW}$)	0
Results	**Optimum 1**	**Optimum 2**	**Optimum 3**
$\mathbf{C_{L0}}$	1.0136	1.0007	1.0419
$\mathbf{C_{L\alpha}}$	5.7881	5.7927	5.9597
$\mathbf{S_{ref}}$ [$\mathbf{m^2}$]	31.3433	31.3433	37.6120
A [$\mathbf{m^2}$]	75.6560	75.6620	89.3698
I	0.0515	0.0465	0.0601

Table 7.7: Parameters and results from first sweep for optimum values

if the lift coefficient performance is slightly better, the skin coefficient area is similar to the case of positive stagger. In this second refinement, negative stagger is included for completeness. This gives a total of 14641 combinations that were all complying with the surface and weight constraints. After simulating them in APAME, the second filter reduces these number to 7859 combinations that exceed the conventional wing performance in terms of C_L. Again the Paretto front from these combinations is plotted in Figure 7.6a and the lowest weight (skin area) points as optimum points. In this case, the point corresponding to the lowest weight region and minimum I^{-1}, which is maximum integral difference between lift coefficient curves, is highlighted and compared to the first three weight minima.

Figure 7.6b is representing the C_L vs α curves for the previously mentioned optima and the quantitative results are summarized in Table 7.9. All error percentages in the results section of the table are expressed relative to optimum 1 values. First relevant fact is that all optima have the same wingspan, MAC and height between wings and only vary in terms of stagger. This is reasonable since minimum weight configuration compliant with the constraints is expected to vary slightly with the stagger. Second relevant fact is the lower overall performance of positive stagger against negative stagger in terms of I, lead by a decrease in C_{L0}. This observation is in line with the trends from the parametric study discussed in the previous section. However, the decrease and increase from optima 2 & 3 has a magnitude of 4% which would not be critical to choose between them. Third relevant fact is the superior performance of optimum 10. The value of I increases a 10% with respect to the value of no stagger with an increase in the skin area of less than 0.03%.

If this optimum is compared with the conventional wing values (Table 7.10), interesting results can be extracted. In the first place, lift coefficient slope and lift coefficient for zero angle of attack remain almost unaffected with changes of ±2%. In terms of the reference surface of the box wing, it was reduced to 6.09 times the reference surface of the conventional wing, and the skin area was reduced 5.65 times compared to its equivalent conventional wing value. This would mean that an equivalent box wing has a superior performance than the conventional wing, allowing both weight savings, lower skin friction drag, and similar lift performances. However, several simplifications were taken into account to reach to this final design and the limits of this assertions need to be discussed in detail.

7.4. LIMITATIONS

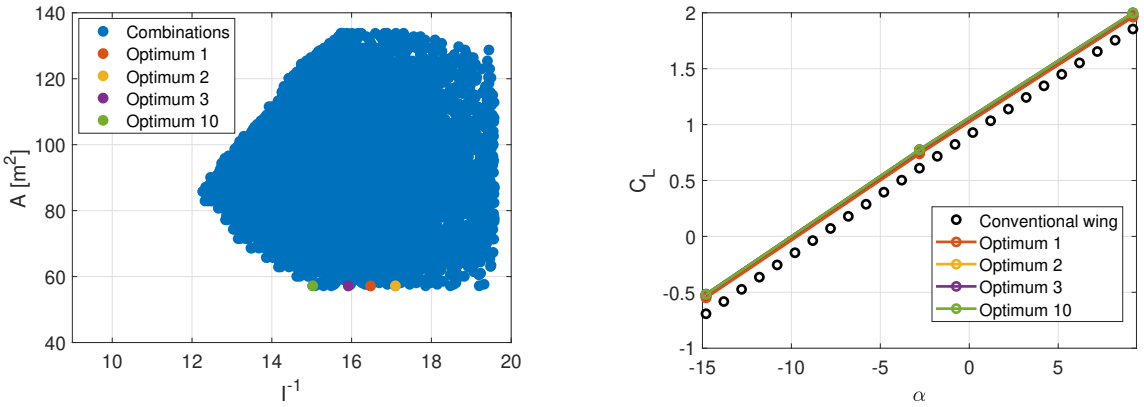
Results of this optimization process need to be analyzed critically and taking into account the inherent limitations that panel methods present. The main important assumptions together with the limitations they are related to are summarized in the following list:

1. The regime of operation of the wings lie in the linear part of the lift coefficient curve and it is steady. On the one hand, non linear effects are expected in reality including stall, which can not be predicted with this model. Note that for the case of box wings the stalling characteristics could be either enhanced or worsened depending on the interaction between upper and lower wings. On the other hand, APAME has proven to be reliable in the linear region in terms of lift, which implies that the performance in terms of C_L of an equivalent box wing would be similar than conventional one at least when both are in the linear regime.

2. The skin friction drag is related to the area covering the skin of the airfoil and the skin material quality. Under the same conditions: same skin surface roughness and attached flow, the box wing would

b [m]	Start: $0.4b_{CW}$ End: $0.6b_{CW}$ Points: 11
MAC [m]	Start: $0.2MAC_{CW}$ End: $0.3MAC_{CW}$ Points: 11
h/b	Start: 0.05 End: 0.15 Points: 11
s/MAC	Start: 1 End: -1 Points: 11
Airfoilwings	MRevE-v2$_{HC}$
Airfoilwinglet	NACA 0030
R [%]	15

Table 7.8: Second sweep on parameters for optimal box wing design



(a) Paretto front

(b) C_L vs α

Figure 7.6: Second sweep for optimum values

Parameters	**Optimum 1**	**Optimum 2**	**Optimum 3**	**Optimum 10**
b [m]	16.9878 ($0.4b_{CW}$)	16.9878 ($0.4b_{CW}$)	16.9878 ($0.4b_{CW}$)	16.9878 ($0.4b_{CW}$)
MAC [m]	0.7085 ($0.2MAC_{CW}$)	0.7085 ($0.2MAC_{CW}$)	0.7085 ($0.2MAC_{CW}$)	0.7085 ($0.2MAC_{CW}$)
h [m]	1.6988 (0.1b)	1.6988 (0.1b)	1.6988 (0.1b)	1.6988 (0.1b)
s [m]	0	0.1417 (0.2MAC)	-0.1417(-0.2MAC)	-0.7085(-MAC)
Results	**Optimum 1**	**Optimum 2**	**Optimum 3**	**Optimum 10**
$\mathbf{C_{L0}}$	1.0404	1.0382 (-0.21%)	1.0495 (0.87%)	1.0617 (2.05%)
$\mathbf{C_{L\alpha}}$	5.9705	5.9720 (0.025%)	5.9747 (0.07%)	6.0125 (0.70%)
$\mathbf{S_{ref}}$ [$\mathbf{m^2}$]	24.0717	24.0717 (0%)	24.0717 (0%)	24.0717 (0%)
A [$\mathbf{m^2}$]	57.1967	57.1974 (0.001%)	57.1976 (0.0015%)	57.2102 (0.024%)
I	0.0607	0.0585 (-3.62%)	0.0628 (3.46%)	0.0666 (9.72%)

Table 7.9: Parameters and results from second sweep for optimum values

present a much lower skin friction drag than the conventional wing (Recap the skin area was reduced 5.65 times). However, whenever the flight conditions are not ideal or in the non linear regime, no statement can be made about the skin friction drag.

3. The form drag is related to the frontal area of the wing that faces the flow. The frontal area of the designed box wing was around five times less than the frontal area of the conventional one at zero angle of attack. This means that under the assumption of attached flow at this α, flow would be rejoined in a

Results	Conventional wing	Box wing
C_{L0}	0.9076	1.0617 (1.70%)
$C_{L\alpha}$	6.0798	6.0125 (-1.11%)
S_{ref} [m^2]	150.4478	24.0717 (-84%)
A [m^2]	323.3616	57.2102 (-82.31%)

Table 7.10: Results comparison between conventional and box wing designs

quicker and smoother way for the case of the box wing. However, if the angle of attack changes, a more detailed computation of the area facing the flow would be needed, together with the interference that the upper wing may induce in the lower one.

4. The weight of the wing was strongly simplied to Eq. 7.10. Recap that this is not taking into account neither differences in materials, nor thicknesses, nor internal structural elements neither in the conventional wing nor in the box wing design. However, the savings in the weight of the box wing skin (84%) give some margin for all these features not being taken into account to be included.

5. The structural deformation of the wing is not taken into account for the box wing case. The conventional wing results are based on a deformed wing geometry, with maximum wing tip displacements of the order of 0.23-1.23% the wingspan of the conventional wing. However, the existence of the winglets joining the upper and lower wings is expected to provide more rigidity to the design and produce less deformations than in the conventional case. The region between the winglets should consider the use of stiffening elements in order to prevent undesired bending of the geometry.

8

CONCLUSIONS

This is a concluding chapter that summarizes the work done, the main results and the future research aligned with the topic.

8.1. CONCLUSIONS

In this project, an automatic framework for box wing low and high fidelity aerodynamic studies was developed. The framework could be used for two different approaches. The first one is for evaluation of an existent box wing design. The second one is for obtaining box wing designs equivalent to conventional wings by means of optimization.

In the first place, the box wing geometry was fully parametrized with eight physical parameters, five airfoil profiles and five discretization parameters. This allowed a wide range of combinations for box wing designs.

In the second place, the geometry was prepared to be meshed for two different aerodynamic tools. First tool is the low fidelity solver, which uses a steady panel method for aerodynamic computations. For this application, a surface mesh with low resolution was shown to provide fast and accurate values of the lift coefficient in the linear region. Second tool is the high fidelity solver, which uses discretized RANS equations, requiring a high quality volume mesh. Among the three options medium or high resolution seem to provide accurate results in terms of lift and drag coefficient in all regions. However, for points in the lift coefficient curve after the maximum lift coefficient, high resolution is strongly recommended.

In the third place, the post processing of the CFD computations show relevant features of box wing aerodynamics such as delayed stall of the upper wing as compared to the lower one, delayed boundary layer separation of the wing tip as compared to the wing root sections and vortex recombination behaviour.

In the last place, an equivalent box wing design to a megawatt conventional wing was performed using the low fidelity tool. An optimization process was executed reducing the reference surface and the area of the skin of the conventional wing an 84% and 82%, respectively. Similar values of lift coefficient slope and lift coefficient at zero angle of attack were obtained during this process.

8.2. FUTURE WORK

The high degree of automation of this work allows to define the start of a wide variety of applications for box wing designs applied to AWE systems. However, since this study was focused on the automation, strong simplifications were made for the aerodynamic analyses and optimization of the low fidelity tool. Among the different options, the most relevant ones for the continuation of this work are listed as follows:

- Develop a detailed structural model that gives a better estimation of the weight and resistance to the aerodynamic loads. This could be coupled with the low fidelity solver to perform fast aero-structural optimization studies.
- Perform CFD studies on the two possible staggered configurations of the validation case and comparison.
- Perform validation and calibration of CFD parameters using wind tunnel experimental data of cambered box wing designs.

- Verify the results obtained for the equivalent megawatt wing of this work with CFD or experiments.
- Increase the complexity of the functions used for the CFD meshing, allowing a higher degree of flexibility in this process.

BIBLIOGRAPHY

[1] International Energy Agency, *World Energy Outlook 2021*. [Online]. Available: https://www.iea.org/reports/world-energy-outlook-2021 (visited on 03/18/2022).

[2] ——, *Renewables 2020, Wind*. [Online]. Available: https://www.iea.org/reports/renewables-2020/wind (visited on 03/18/2022).

[3] C. Jung and D. Schindler, "A global wind farm potential index to increase energy yields and accessibility," *Energy*, vol. 231, p. 120 923, 2021, ISSN: 0360-5442. DOI: 10.1016/j.energy.2021.120923.

[4] P. Enevoldsen and G. Xydis, "Examining the trends of 35 years growth of key wind turbine components," *Energy for Sustainable Development*, vol. 50, pp. 18–26, 2019, ISSN: 0973-0826. DOI: 10.1016/j.esd.2019.02.003.

[5] R. Brood, Ö. Ceyhan, W. Engels, J. Peeringa, and G. De Winkel, "Upwind 20mw wind turbine pre-design: Blade design and control," ECN-E-11-017, Tech. Rep., 2011.

[6] M. Diehl, "Airborne wind energy: Basic concepts and physical foundations," in *Airborne Wind Energy*, U. Ahrens, M. Diehl, and R. Schmehl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 3–22, ISBN: 978-3-642-39965-7. DOI: 10.1007/978-3-642-39965-7_1.

[7] A. Cherubini, A. Papini, R. Vertechy, and M. Fontana, "Airborne wind energy systems: A review of the technologies," *Renewable and Sustainable Energy Reviews*, vol. 51, pp. 1461–1476, 2015.

[8] R. Schmehl, *Airborne wind energy – An introduction to an emerging technology*. [Online]. Available: http://www.awesco.eu/awe-explained/ (visited on 03/21/2022).

[9] S. Watson, A. Moro, V. Reis, *et al.*, "Future emerging technologies in the wind power sector: A european perspective," *Renewable and sustainable energy reviews*, vol. 113, pp. 109–270, 2019.

[10] A. Cherubini, G. Moretti, and M. Fontana, "Dynamic modeling of floating offshore airborne wind energy converters," in *Airborne Wind Energy: Advances in Technology Development and Research*, R. Schmehl, Ed. Singapore: Springer Singapore, 2018, pp. 137–163, ISBN: 978-981-10-1947-0. DOI: 10.1007/978-981-10-1947-0_7.

[11] European Commission and Directorate-General for Research and Innovation, *Study on challenges in the commercialisation of airborne wind energy systems*. Publications Office, 2018. DOI: 10.2777/933106.

[12] G. Lutsch, "Airborne Wind Energy Network HWN500–Shouldering R&D in Co-Operations," in *Book of Abstracts AWEC 2015*, AWEC, 2015.

[13] Makani Power, *The Energy Kite, Selected Results From the Design, Development and Testing of Makani's Airborne Wind Turbines Part I*, 2020. [Online]. Available: https://airbornewindeurope.org/wp-content/uploads/2022/04/Makani-2020_TheEnergyKiteReport_Part1-web.pdf (visited on 05/26/2022).

[14] Ampyx Power, *Technology: Demonstrator AP3*, 2017. [Online]. Available: https://www.ampyxpower.com/technology/demonstrator-ap3/ (visited on 05/26/2022).

[15] R. B. Addoms, "Aerodynamic and structural design considerations for high lift biplane wing systems," Ph.D. dissertation, University of California, Los Angeles, 1972.

[16] R. B. Addoms and F. W. Spaid, "Aerodynamic design of high-performance biplane wings," *Journal of Aircraft*, vol. 12, no. 8, pp. 629–630, 1975. DOI: 10.2514/3.59846.

[17] E. Olson and B. Selberg, "Experimental determination of improved aerodynamic characteristics utilizing biplane wing configurations," *Journal of Aircraft*, vol. 13, no. 4, pp. 256–261, 1976. DOI: 10.2514/3.44523.

[18] M. L. Loyd, "Crosswind kite power (for large-scale wind power production)," *Journal of Energy*, vol. 4, no. 3, pp. 106–111, 1980. DOI: 10.2514/3.48021.

[19] S. Costello, C. Costello, G. François, and D. Bonvin, "Analysis of the maximum efficiency of kite-power systems," *Journal of Renewable and Sustainable Energy*, vol. 7, no. 5, pp. 53–108, 2015. DOI: 10.1063/1.4931111.

[20] L. Prandtl, *Induced drag of multiplanes (technical note No. 182)*, 1924.

[21] P. D. Gall and H. C. Smith, "Aerodynamic characteristics of biplanes with winglets," *Journal of Aircraft*, vol. 24, no. 8, pp. 518–522, 1987. DOI: 10.2514/3.45470.

[22] S. A. Andrews, "Multidisciplinary analysis of closed, nonplanar wing configurations for transport aircraft," Ph.D. dissertation, Royal Military College of Canada, 2016.

[23] J. Bevirt, *Tethered airborne power generation system with vertical take-off and landing capability*, US Patent App. 12/784,328, Nov. 2010.

[24] KiteKraft, *KiteKraft Technology*. [Online]. Available: https://www.kitekraft.de/technology (visited on 02/22/2022).

[25] ——, *KiteKraft Blog*. [Online]. Available: https://medium.com/kitekraft/kitekraft-succeeds-with-autonomous-all-phase-flight-of-new-kite-demonstrator-f13b12690d55 (visited on 02/22/2022).

[26] F. A. Khan, "Preliminary aerodynamic investigation of box-wing configurations using low fidelity codes," M.S. thesis, Luleå University of Technology Dept. of Space Science, Kiruna, 2010.

[27] H. Gagnon and D. W. Zingg, "Aerodynamic optimization trade study of a box-wing aircraft configuration," *Journal of Aircraft*, vol. 53, no. 4, pp. 971–981, 2016. DOI: 10.2514/1.C033592.

[28] S. A. Andrews and R. E. Perez, "Comparison of box-wing and conventional aircraft mission performance using multidisciplinary analysis and optimization," *Aerospace Science and Technology*, vol. 79, pp. 336–351, 2018, ISSN: 1270-9638. DOI: 10.1016/j.ast.2018.05.060.

[29] F. Bauer, R. M. Kennel, C. M. Hackl, F. Campagnolo, M. Patt, and R. Schmehl, "Drag power kite with very high lift coefficient," *Renewable Energy*, vol. 118, pp. 290–305, 2018, ISSN: 0960-1481. DOI: 10.1016/j.renene.2017.10.073.

[30] U. Zillmann and P. Bechtle, "Emergence and economic dimension of airborne wind energy," in *Airborne Wind Energy: Advances in Technology Development and Research*, R. Schmehl, Ed. Singapore: Springer Singapore, 2018, pp. 1–25, ISBN: 978-981-10-1947-0. DOI: 10.1007/978-981-10-1947-0_1.

[31] H. Schmidt, G. de Vries, R. Schmehl, and R. Renes, "The social acceptance of airborne wind energy: A literature review," *Energies*, vol. 15, no. 4, 2022, ISSN: 1996-1073. DOI: 10.3390/en15041384.

[32] IRENA, *Offshore renewables: An action agenda for deployment, International Renewable Energy Agency*. Abu Dhabi, 2021, ISBN: 978-92-9260-349-6.

[33] United Nations Climate Change Technology Executive (UNCCTE) Committee, *Emerging climate technologies in the energy supply sector*. [Online]. Available: https://unfccc.int/ttclear/tec/energysupplysector.html (visited on 03/07/2022).

[34] J. Weber, M. Marquis, A. Cooperman, *et al.*, "Airborne wind energy," National Renewable Energy Lab (NREL), Golden, CO (United States), Tech. Rep., 2021.

[35] V. Salma, R. Ruiterkamp, M. Kruijff, M. M. R. van Paassen, and R. Schmehl, "Current and expected airspace regulations for airborne wind energy systems," in *Airborne Wind Energy: Advances in Technology Development and Research*, R. Schmehl, Ed. Singapore: Springer Singapore, 2018, pp. 703–725, ISBN: 978-981-10-1947-0. DOI: 10.1007/978-981-10-1947-0_29.

[36] F. A. Administration, *Notification for Airborne Wind Energy Systems (AWES), FAA-2011-1279, Dec 2011*. [Online]. Available: https://www.govinfo.gov/content/pkg/FR-2011-12-07/pdf/2011-31430.pdf (visited on 03/17/2022).

[37] Daniel Filkovic, *APAME-Aircraft 3D panel method*, 2010. [Online]. Available: http://www.3dpanelmethod.com/home.html (visited on 05/27/2022).

[38] Cadence, *Pointwise for cfd meshing*. [Online]. Available: https://www.pointwise.com/ (visited on 05/24/2022).

[39] OpenFOAM, *About openfoam*. [Online]. Available: https://www.openfoam.com/ (visited on 05/24/2022).

8

[40] Mathworks, *Matlab*. [Online]. Available: https://www.mathworks.com/products/matlab.html (visited on 05/24/2022).

[41] ParaView, *Welcome to paraview*. [Online]. Available: https://www.paraview.org/ (visited on 05/24/2022).

[42] Pointwise, *Glyph version 3.18.6*. [Online]. Available: https://www.pointwise.com/glyph2/files/Glyph/cxx/GgGlyph-cxx.html (visited on 05/24/2022).

[43] J. Katz and A. Plotkin, *Low-Speed Aerodynamics*, 2nd ed., ser. Cambridge Aerospace Series. Cambridge University Press, 2001. DOI: 10.1017/CBO9780511810329.

[44] D. Eijkelhof, "Design and optimisation framework of a multi-MW airborne wind energy reference system," M.S. thesis, Delft University of Technology & Technical University of Denmark, 2019.

[45] U. Fasel, P. Tiso, D. Keidel, G. Molinari, and P. Ermanni, "Reduced-order dynamic model of a morphing airborne wind energy aircraft," *AIAA Journal*, vol. 57, no. 8, pp. 3586–3598, 2019. DOI: 10.2514/1.J058019.

[46] C. Hirsch, "Basic discretization techniques," in *Numerical Computation of Internal and External Flows (Second Edition)*, C. Hirsch, Ed., Second Edition, Oxford: Butterworth-Heinemann, 2007, pp. 141–144, ISBN: 978-0-7506-6594-0. DOI: 10.1016/B978-075066594-0/50044-8.

[47] O. Kolditz, "Finite volume method," in *Computational Methods in Environmental Fluid Mechanics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 173–190, ISBN: 978-3-662-04761-3. DOI: 10.1007/978-3-662-04761-3_8.

[48] J. Boussinesq, *Essai sur la théorie des eaux courantes*. Impr. nationale, 1877.

[49] L. Prandtl, "7. bericht über untersuchungen zur ausgebildeten turbulenz," *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 5, no. 2, pp. 136–139, 1925. DOI: 10.1002/zamm.19250050212.

[50] A. N. Kolmogorov, "Equations of turbulent motion in an incompressible fluid," in *Dokl. Akad. Nauk SSSR*, vol. 30, 1941, pp. 299–303.

[51] L. Prandtl, "Uber ein neues formel-system fur die ausgebildete turbulenz, nachr akad wiss, gottingen, math," *Phys. Kl*, vol. 1945, p. 6, 1945.

[52] P. SPALART and S. ALLMARAS, "A one-equation turbulence model for aerodynamic flows," in *30th Aerospace Sciences Meeting and Exhibit*. DOI: 10.2514/6.1992-439.

[53] P. Catalano and M. Amato, "An evaluation of rans turbulence modelling for aerodynamic applications," *Aerospace Science and Technology*, vol. 7, no. 7, pp. 493–509, 2003, ISSN: 1270-9638. DOI: https://doi.org/10.1016/S1270-9638(03)00061-0.

[54] W. Jones and B. Launder, "The prediction of laminarization with a two-equation model of turbulence," *International Journal of Heat and Mass Transfer*, vol. 15, no. 2, pp. 301–314, 1972, ISSN: 0017-9310. DOI: 10.1016/0017-9310(72)90076-2.

[55] B. Launder and D. Spalding, "Paper 8 - the numerical computation of turbulent flows," in *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion*, S. V. Patankar, A. Pollard, A. K. Singhal, and S. P. Vanka, Eds., Pergamon, 1983, pp. 96–116, ISBN: 978-0-08-030937-8. DOI: 10.1016/B978-0-08-030937-8.50016-7.

[56] D. Wilcox, *Turbulence Modeling for CFD*, 3rd ed. DCW Industries, 2006, vol. 1, ISBN: 9781928729082.

[57] F. Menter, "Zonal two equation k-w turbulence models for aerodynamic flows," in *23rd Fluid Dynamics, Plasmadynamics, and Lasers Conference*. DOI: 10.2514/6.1993-2906.

[58] F. R. Menter, M. Kuntz, and R. Langtry, "Ten years of industrial experience with the sst turbulence model," in *International Symposium on Turbulence, Heat and Mass Transfer*, vol. 4, West Redding: Begell House Inc., 2003, pp. 625–632.

[59] L. S. Caretto, A. D. Gosman, S. V. Patankar, and D. B. Spalding, "Two calculation procedures for steady, three-dimensional flows with recirculation," in *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, H. Cabannes and R. Temam, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1973, pp. 60–68, ISBN: 978-3-540-38392-5. DOI: 10.1007/BFb0112677.

[60] OpenFOAM, *OpenFOAM: User Guide v2112. SIMPLE algorithm.* [Online]. Available: https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-simple.html (visited on 05/16/2022).

[61] H. Jasak, "Error analysis and estimation for the finite volume method with applications to fluid flows.," Ph.D. dissertation, Imperial College London (University of London), 1996.

[62] J. P. Van Doormaal and G. D. Raithby, "Enhancements of the simple method for predicting incompressible fluid flows," *Numerical heat transfer*, vol. 7, no. 2, pp. 147–163, 1984.

[63] H. Schlichting and K. Gersten, "Internal flows," in *Boundary-Layer Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 519–556, ISBN: 978-3-662-52919-5. DOI: 10.1007/978-3-662-52919-5_17. [Online]. Available: https://doi.org/10.1007/978-3-662-52919-5_17.

[64] J. Chawner, *Quality and Control – Two Reasons Why Structured Grids Aren't Going Away.* [Online]. Available: https://www.pointwise.com/articles/quality-and-control-two-reasons-why-structured-grids-aren-t-going-away (visited on 05/24/2022).

[65] Pointwise, *Area and Volume Ratio.* [Online]. Available: https://www.pointwise.com/doc/user-manual/examine/functions/area-volume-ratio.html#description (visited on 05/24/2022).

[66] ——, *Equiangle Skewness.* [Online]. Available: https://www.pointwise.com/doc/user-manual/examine/functions/equiangle-skewness.html (visited on 05/24/2022).

[67] SIMSCALE, *Documentation. cfd numerics: Non-orthogonal correctors.* [Online]. Available: https://www.simscale.com/docs/simulation-setup/numerics/non-orthogonal-correctors/ (visited on 05/30/2022).

[68] CFD Direct, *Openfoam v7 user guide: 4.5 numerical schemes.* [Online]. Available: https://cfd.direct/openfoam/user-guide/v7-fvschemes/ (visited on 05/30/2022).

[69] Pointwise, *Centroid skewness.* [Online]. Available: https://www.pointwise.com/doc/user-manual/examine/functions/centroid-skewness.html (visited on 05/30/2022).

[70] ——, *Aspect ratio.* [Online]. Available: http://www.pointwise.com/doc/user-manual/examine/functions/aspect-ratio.html (visited on 05/30/2022).

[71] ——, *T-rex tab.* [Online]. Available: https://www.pointwise.com/doc/user-manual/grid/solve/unstructured-domains/t-rex.html (visited on 05/24/2022).

[72] M. Athadkar and S. Desai, "Importance of the extent of far-field boundaries and of the grid topology in the cfd simulation of external flows," *International Journal of Mechanical And Production Engineering*, vol. 2, pp. 69–72, 2014, ISSN: 2320-2092.

[73] OpenFOAM, *Openfoam user guide: 3.2 running applications in parallel.* [Online]. Available: https://www.openfoam.com/documentation/user-guide/3-running-applications/3.2-running-applications-in-parallel (visited on 05/31/2022).

[74] CFD Online, *Turbulence intensity.* [Online]. Available: https://www.cfd-online.com/Wiki/Turbulence_intensity (visited on 05/31/2022).

[75] ——, *Eddy viscosity ratio.* [Online]. Available: https://www.cfd-online.com/Wiki/Eddy_viscosity_ratio (visited on 05/31/2022).

[76] G. Lebesque, "Steady-state rans simulation of a leading edge inflatable wing with chordwise struts," M.S. thesis, Delft University of Technology, 2020.

[77] OpenFOAM, *Openfoam v8 user guide: 4.5 numerical schemes.* [Online]. Available: https://cfd.direct/openfoam/user-guide/v8-fvschemes/ (visited on 05/31/2022).

[78] L. F. Richardson and J. A. Gaunt, "Viii. the deferred approach to the limit," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 226, no. 636-646, pp. 299–361, 1927. DOI: 10.1098/rsta.1927.0008.

[79] I. H. Abbott and A. E. Von Doenhoff, *Theory of wing sections: Including a summary of airfoil data.* Dover Publications, Inc., New York, 1959.

[80] D. J. Acheson, *Elementary fluid dynamics: Oxford University Press.* Oxford, England, 1990.

[81] D. E. Gault, "A correlation of low-speed, airfoil-section stalling characteristics with reynolds number and airfoil geometry," Tech. Rep. NACA-TN-3963, 1957.

8

[82] A. Haines, “Scale effects on aircraft and weapon aerodynamics (les effets d’echelle et l’aerodynamique des aeronefs et des systemes d’armes),” Advisory Group For Aerospace Research and Development (AGARD), Neuilly-Sur-Seine, France, Tech. Rep. ADA291964, 1994.

A

GEOMETRIC COMPUTATIONS

A.1. CENTER OF ROTATION

In order to compute the center of rotation between two points one located at the wing and the other at the winglet, Figure A.1 serves as illustration of the following steps being followed:

- Vectors $\vec{r}_1 = \overline{OA}$ and $\vec{r}_2 = \overline{OB}$ are defined. Note that the position of the center of rotation O is unknown and these two vectors are thus a function of this position.
- Vectors $\vec{h} = [s\ 0\ h]$ and its normal vector $\vec{n} = [h\ 0\ -s]$ contained in the plane xz are defined and normalized.
- In order to find the position of the center of rotation, the following equations are formulated:
  - Since the turn must be 90 deg both vectors must be perpendicular $\vec{r}_1 \cdot \vec{r}_2 = 0$
  - Since the arc between points A and B must be circular then $|\vec{r}_1| = |\vec{r}_2|$
  - Since the plane containing the arc must be perpendicular to $\vec{h}$ then $\frac{\vec{r}_1 \times \vec{r}_2}{|\vec{r}_1 \times \vec{r}_2|} = \vec{n}$
- Finally these equations are solved and the position of O is obtained.

A.2. ELEMENT SIZE IN A CURVE

In order to compute the element size in the last layer of the boundary layer, some steps must be followed. Figure A.2 aims to illustrate the variables involved in the process.

- The spacing in the vertical direction of the boundary layer (s_v) is strongly dependent on the layer number (n). It is also influenced by the growth rate (GR) and the initial boundary layer thickness (y_i).

$$s_v(n) = y_i * GR^n \tag{A.1}$$

- The angle between two lines perpendicular the curve (α) is computed by specifying the desired divisions in the curved region (N_{div}) and thus the surface mesh size (l_c).

$$l_c = \frac{\pi R}{2N_{div}} \tag{A.2}$$

$$\alpha = 90/N_{div} \tag{A.3}$$

- The radius of the curve (R) is known and the vectors giving the position in the curve $\vec{r}_1(n) = [x_1(n), y_1(n)]$ and $\vec{r}_2(n) = [x_2(n), y_2(n)]$ are generated as a function of the layer number following these equations:

$$x_1(n) = R - s_v(n) \tag{A.4}$$

$$y_1(n) = 0 \tag{A.5}$$

$$x_2(n) = (R - s_v(n))\cos\alpha \tag{A.6}$$

$$y_2(n) = (R - s_v(n))\sin\alpha \tag{A.7}$$

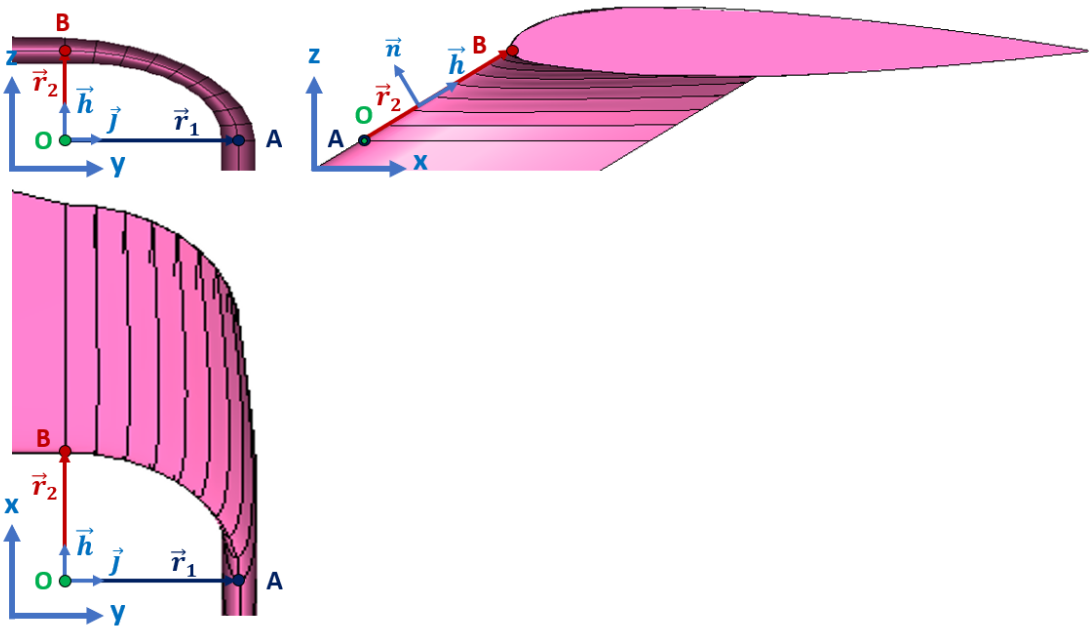


Figure A.1: Graphic representation of the center of rotation taking as an example the points on the LE

- The element size as a function of the layer number is computed as:

$$\left|\vec{d}\,(n)\right| = |\vec{r}_2\,(n) - \vec{r}_1\,(n)| \tag{A.8}$$

- Finally, the surface element size (l_s) in the straight region is just the distance between two curves in the last layer (N_{max})

$$l_s = \left|\vec{d}\,(N_{max})\right| \tag{A.9}$$

However, if this process is used for 3D meshes, a correction factor ($F \leq 1$) need to be applied to take into account 3D effects.

$$l_s = F\left|\vec{d}\,(N_{max})\right| \tag{A.10}$$

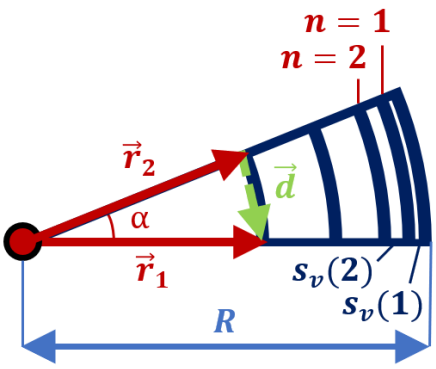


Figure A.2: Graphic representation of the element size computation in a curve

B

CODE

All the code with some example scripts can be found on GitHub:
https://github.com/gabrielebuendiavela/aerABox.

B.1. BOX WING PARAMETRIZATION

```
% This function generates the geometry of a boxwing given certain
% parameters and stores it in two different ways: a matrix containing all
% the cross sections defining the wing and a list of all points present
% in the wing
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       h         -> Height between wings [m]
%       b         -> Wingspan [m]
%       R         -> Cant radius [m]
%       c_r1      -> Chord at the root of the lower wing [m]
%       c_t1      -> Chord at the tip of the lower wing [m]
%       c_r2      -> Chord at the root of the upper wing [m]
%       c_t2      -> Chord at the tip of the upper wing [m]
%       s         -> Wing stagger [m]
%       N_bu      -> Number of spanwise sections upper wing
%       N_bl      -> Number of spanwise sections lower wing
%       N_bw      -> Number of spanwise sections winglet
%       N_el      -> Number of elements to discretize airfoils
%       N         -> Number of sections to discretize the cant region
%       Finite    -> Finite trailing edge: 1-yes | 0-no
%       af_1r     -> Airfoil at the root of the lower wing
%       af_1t     -> Airfoil at the tip of the lower wing
%       af_2r     -> Airfoil at the root of the upper wing
%       af_2t     -> Airfoil at the tip of the upper wing
%       af_w      -> Airfoil at the winglet
% Outputs:
%       Slices_Wing -> Matrix [N_points_airfoil x 3 x N_slices] containing
%                         all the wing cross sections [m]
%       Points_Wing -> Matrix [N_points_airfoil*N_slices x 3] containing
%                         all the points of the wing [m]
function [Slices_Wing, Points_Wing] = ...
    ParametrizationBWFiniteTE(h,b,R,c_r1,c_r2,c_t1,c_t2,s,N_bu,N_bl,N_bw,N_el,N, ...
    Finite,af_1r,af_2r,af_1t,af_2t,af_w)
addpath("sr_bwp\")

% Define the airfoils
% load("MEGAWESAirfoil.mat") % You can choose to load your own airfoil geometry, then ...
    you should specify it manually at each section
airfoil_1root = generateAirfoil(af_1r,length(af_1r),N_el,Finite);   % Dimensionless ...
    airfoil profile 1 root
% airfoil_1root = AirfoilRevEHC; % Replace your own airfoil variable name
airfoil_1root = swapmatrix(airfoil_1root);
```

```
airfoil_1tip = generateAirfoil(af_1t,length(af_1t),N_el,Finite);    % Dimensionless ...
    airfoil profile 1 tip
% airfoil_1tip = AirfoilRevEHC; % Replace your own airfoil variable name
airfoil_1tip = swapmatrix(airfoil_1tip);
airfoil_2root = generateAirfoil(af_2r,length(af_2r),N_el,Finite);  % Dimensionless ...
    airfoil profile 2 root
% airfoil_2root = AirfoilRevEHC; % Replace your own airfoil variable name
airfoil_2tip = generateAirfoil(af_2t,length(af_2t),N_el,Finite);    % Dimensionless ...
    airfoil profile 2 tip
% airfoil_2tip = AirfoilRevEHC; % Replace your own airfoil variable name
airfoil_winglet = generateAirfoil(af_w,length(af_w),N_el,Finite); % Dimensionless ...
    airfoil profile winglet
% airfoil_winglet = generateAirfoil(af_w,length(af_w),N_el,Finite); % Replace your ...
    own airfoil variable name

% Close open profiles (finite TE)
if Finite == 1
    prov_size = length(airfoil_winglet);
    airfoil_1root(prov_size+1,:) = airfoil_1root(1,:);
    airfoil_1tip(prov_size+1,:) = airfoil_1tip(1,:);
    airfoil_2root(prov_size+1,:) = airfoil_2root(1,:);
    airfoil_2tip(prov_size+1,:) = airfoil_2tip(1,:);
    airfoil_winglet(prov_size+1,:) = airfoil_winglet(1,:);
else
    airfoil_1root(1,2) = 0; airfoil_1root(end,2) = 0;
    airfoil_1tip(1,2) = 0; airfoil_1tip(end,2) = 0;
    airfoil_2root(1,2) = 0; airfoil_2root(end,2) = 0;
    airfoil_2tip(1,2) = 0; airfoil_2tip(end,2) = 0;
    airfoil_winglet(1,2) = 0; airfoil_winglet(end,2) = 0;
end

% Calculations
airfoil_1r = c_r1*airfoil_1root;            % Dimensional airfoil 1 at the root
airfoil_1t = c_t1*airfoil_1tip;             % Dimensional airfoil 1 at the tip
airfoil_2r = c_r2*airfoil_2root;            % Dimensional airfoil 2 at the root
airfoil_2t = c_t2*airfoil_2tip;             % Dimensional airfoil 2 at the tip
airfoil_1w = c_t1*airfoil_winglet;          % Dimensional airfoil winglet
airfoil_2w = c_t2*airfoil_winglet;          % Dimensional airfoil winglet
vec_h = [s 0 h]; vec_h = vec_h/norm(vec_h); % Versor connecting airfoils 1 and 2
vec_hn = [vec_h(3) 0 -vec_h(1)];            % Versor normal to vec_h
vec_i = [1 0 0];                            % Versor from x axis
vec_j = [0 1 0];                            % Versor from y axis
vec_k = [0 0 1];                            % Versor from z axis
p_O = [0 0 0];                              % Origin
p_A = [0 b/2-R 0];                          % End of lower wing
p_B = p_A + R*vec_h;                        % Center of circunference 1
p_C = p_B + R*vec_j;                        % Lower part of winglet
p_D = p_C + (sqrt(h^2+s^2)-2*R)*vec_h;      % Upper part of winglet
p_E = p_D - R*vec_j;                        % Center of circumference 2
p_F = p_E + R*vec_h;                        % End of upper wing
p_G = p_F - (b/2-R)*vec_j;                  % Root of upper wing

airfoil_O = zeros(length(airfoil_1r),3);
airfoil_O(:,1) = airfoil_1r(:,1); airfoil_O(:,3) = airfoil_1r(:,2);

airfoil_A = zeros(length(airfoil_1t),3)+p_A;
airfoil_A(:,1) = airfoil_1t(:,1)+p_A(1); airfoil_A(:,3) = airfoil_1t(:,2)+p_A(3);

airfoil_C = zeros(length(airfoil_1t),3)+p_C;
airfoil_C(:,1) = airfoil_1w(:,1)+p_C(1); airfoil_C(:,2) = airfoil_1w(:,2)+p_C(2);

airfoil_D = zeros(length(airfoil_2t),3)+p_D;
airfoil_D(:,1) = airfoil_2w(:,1)+p_D(1); airfoil_D(:,2) = airfoil_2w(:,2)+p_C(2);

airfoil_F = zeros(length(airfoil_2t),3)+p_F;
airfoil_F(:,1) = airfoil_2t(:,1)+p_F(1); airfoil_F(:,3) = airfoil_2t(:,2)+p_F(3);

airfoil_G = zeros(length(airfoil_2t),3)+p_G;
airfoil_G(:,1) = airfoil_2r(:,1)+p_G(1); airfoil_G(:,3) = airfoil_2r(:,2)+p_G(3);

center_temp = airfoil_A(1,:)+R*vec_h;
```

B

```
for i = 1:length(airfoil_1t)
    opts = optimoptions('fsolve', 'TolFun', 1E-10, 'TolX', 1E-10);
    center = fsolve(@(r0) ...
        functioncenter(r0,airfoil_A(i,:),airfoil_C(i,:),vec_hn),center_temp,opts);
    center_temp = center;
    CP1(:,:,i) = circunferencearc(center,airfoil_A(i,:),airfoil_C(i,:),N);
end
CP1(:,:,end) = CP1(:,:,1);
center_temp = airfoil_F(1,:)-R*vec_h;
for i = 1:length(airfoil_2t)
    opts = optimoptions('fsolve', 'TolFun', 1E-10, 'TolX', 1E-10);
    center = fsolve(@(r0) ...
        functioncenter(r0,airfoil_D(i,:),airfoil_F(i,:),vec_hn),center_temp,opts);
    center_temp = center;
    CP2(:,:,i) = circunferencearc(center,airfoil_D(i,:),airfoil_F(i,:),N);
end
CP2(:,:,end) = CP2(:,:,1);

% Join points and organize in lines
N_lines = length(airfoil_1r);
N_points_lines = 2*(N-2)+6+N_bu+N_bw+N_bl;
Points_Wing = zeros(N_lines*N_points_lines,3);
k = 0;
% Optional figure, suppress in optimization applications
% Uncomment lines 129, 130, 157, 193-199 for visualization
% figure()
% hold on
for i = 1:N_lines
        counter = i+(N_points_lines-1)*k;
        Points_Wing(counter,:) = airfoil_O(i,:);
        aux_low = (airfoil_A(i,:)-airfoil_O(i,:))/(N_bl+1);
        for j  = 1:N_bl
            Points_Wing(counter+j,:) = airfoil_O(i,:)+j*aux_low;
        end
        Points_Wing(counter+1+N_bl,:) = airfoil_A(i,:);
        for j = 1:(N-2)
            Points_Wing(counter+1+N_bl+j,:) = CP1(j,:,i);
        end
        Points_Wing(counter+N+N_bl,:) = airfoil_C(i,:);
        aux_winglet = (airfoil_D(i,:)-airfoil_C(i,:))/(N_bw+1);
        for j  = 1:N_bw
            Points_Wing(counter+N+N_bl+j,:) = airfoil_C(i,:)+j*aux_winglet;
        end
        Points_Wing(counter+N+N_bl+N_bw+1,:) = airfoil_D(i,:);
        for j = 1:(N-2)
            Points_Wing(counter+j+N+N_bl+N_bw+1,:) = CP2(j,:,i);
        end
       Points_Wing(counter+2*N+N_bl+N_bw,:) = airfoil_F(i,:);
       aux_up = (airfoil_G(i,:)-airfoil_F(i,:))/(N_bu+1);
       for j  = 1:N_bu
           Points_Wing(counter+2*N+N_bl+N_bw+j,:) = airfoil_F(i,:)+j*aux_up;
       end
       Points_Wing(counter+2*N+N_bl+N_bw+N_bu+1,:) = airfoil_G(i,:);
%        plot3(Points_Wing(counter:counter+2*N+N_bl+N_bw+N_bu+1,1), ...
    Points_Wing(counter:counter+2*N+N_bl+N_bw+N_bu+1,2), ...
    Points_Wing(counter:counter+2*N+N_bl+N_bw+N_bu+1,3),'-k','LineWidth',0.5)
       k = k+1;
end

% Join points and organize in slices
N_slices = 2*(N-2)+6+N_bu+N_bl+N_bw;
N_points_slices = length(airfoil_A);
Slices_Wing = zeros(N_points_slices,3,N_slices);
k = 0;
Slices_Wing(:,:,1) = airfoil_O;
Increment_Low = (airfoil_A-airfoil_O)/(N_bl+1);
for i = 1:N_bl
    Slices_Wing(:,:,i+1) = airfoil_O+i*Increment_Low;
end
Slices_Wing(:,:,2+N_bl) = airfoil_A;
for i = 1:N-2
```

```
    aux = CP1(i,:,:);
    Slices_Wing(:,:,i+2+N_bl) = reshape(aux,[3 N_points_slices])';
end
Slices_Wing(:,:,N+1+N_bl) = airfoil_C;
Increment_Winglet = (airfoil_D-airfoil_C)/(N_bw+1);
for i = 1:N_bw
    Slices_Wing(:,:,i+N+1+N_bl) = airfoil_C+i*Increment_Winglet;
end
Slices_Wing(:,:,N+2+N_bl+N_bw) = airfoil_D;
for i = 1:N-2
    aux = CP2(i,:,:);
    Slices_Wing(:,:,N+i+2+N_bl+N_bw) =  reshape(aux,[3 N_points_slices])';
end
Slices_Wing(:,:,2*N+1+N_bl+N_bw) = airfoil_F;
Increment_Up = (airfoil_G-airfoil_F)/(N_bu+1);
for i = 1:N_bu
    Slices_Wing(:,:,i+2*N+1+N_bl+N_bw) = airfoil_F+i*Increment_Up;
end
Slices_Wing(:,:,2*N+2+N_bl+N_bw+N_bu) = airfoil_G;

% for i = 1:N_slices
%     plot3(Slices_Wing(:,1,i),Slices_Wing(:,2,i),Slices_Wing(:,3,i),'b','LineWidth',1.5)
% end
% grid on
% campos([-5.81744979365763,-7.348383400029182,6.328819187324961])
% drawnow
% axis equal
```

B.1.1. AIRFOIL GENERATION

```
% This function generates a 4 or 5 series NACA airfoil
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       designation -> Airfoil designation
%       digits      -> 4 or 5
%       panels      -> Number of points discretizing the airfoil
%       TE          -> Trailing edge
% Outputs:
%       airfoil     -> vector [panels x 2] containing the x and z
%                      coordinates of the airfoil
function airfoil = generateAirfoil(designation,digits,panels,TE)
iaf.designation= designation;
iaf.n= panels;
iaf.HalfCosineSpacing=1;
iaf.wantFile=0;
iaf.datFilePath='./';
iaf.is_finiteTE=TE;
switch digits
    case 4
        af = naca4gen(iaf);
    case 5
        af = naca5gen(iaf);
end
airfoil = zeros(iaf.n*2+1,2);
airfoil(:,1) = af.x;
airfoil(:,2) = af.z;
end
```

B.1.2. CENTER OF ROTATION

```
% This function generates the system of equations to be solved for
% computing the center of rotation
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
```

B

```
%       r0 -> Center of rotation
%       r1 -> Point 1 belonging to the arc
%       r2 -> Point 2 belonging to the arc
%       hn -> Vector normal to the circle plane
% Outputs:
%       F  -> System of five equations equated to 0
function F = functioncenter(r0,r1,r2,hn)
v1 = r1-r0;
v2 = r2-r0;
F(1) = dot(v1,v2);
F(2) = norm(v1)-norm(v2);
aux = cross(v1,v2)/norm(cross(v1,v2));
F(3) = aux(1)-hn(1);
F(4) = aux(2)-hn(2);
F(5) = aux(3)-hn(3);
end
```

B.1.3. CIRCUNFERENCE ARC

```
% Function that computes N points of the arc of the circunference
% given 2 orthogonal points and the center (Note that it only works
% if both points are orthogonal). Note that the arc ends are excluded.
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       rc -> Center of the circunference
%       r1 -> Point 1
%       r2 -> Point 2
%       N  -> Number of points
% Output:
%       CP -> Points on the circunference
function CP = circunferencearc(rc,r1,r2,N)
theta = linspace(0,pi/2,N);
theta = theta(2:end-1);
x = @(t) rc(1) + cos(t).*(r1(1)-rc(1)) + sin(t).*(r2(1)-rc(1));
y = @(t) rc(2) + cos(t).*(r1(2)-rc(2)) + sin(t).*(r2(2)-rc(2));
z = @(t) rc(3) + cos(t).*(r1(3)-rc(3)) + sin(t).*(r2(3)-rc(3));
CP(:,1) = x(theta);
CP(:,2) = y(theta);
CP(:,3) = z(theta);
end
```

B.1.4. SWAP MATRIX

```
% This function swaps the elements of a matrix
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       A -> Matrix to swap
% Outputs:
%       A -> Swapped matrix
function A = swapmatrix(A)
[a,~] = size(A);
b = mod(a,2);
if b == 0
    a = a/2;
else
    a = (a-1)/2;
end
for i = 1:a
    aux = A(i,:);
    A(i,:) = A(end-(i-1),:);
    A(end-(i-1),:) = aux;
end
```

B.2. BOX WING APAME


```
% This function generates an APAME geometry and the input file given
% certain parameters
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       Slices_Wing -> Matrix [N_points_airfoil x 3 x N_slices] containing
%                      all the wing cross sections [m]
%       h           -> Height between wings [m]
%       name        -> Name of the input file
%       airspeed    -> Freestream reference speed [m/s]
%       density     -> Freestream density [kg/m^3]
%       pressure    -> Freestream pressure [Pa]
%       mach        -> Fresstream Mach number
%       cases       -> Vector [n x 2] of cases to simulate in APAME
%                      containing the AoA [deg] and AoS [deg] to simulate
%       wingspan    -> Wingspan [m]
%       MAC         -> Mean aerodynamic chord [m]
%       surf        -> Surface area [m]
%       origin      -> Origin of coordinates [m]
%       method      -> # singularity method:
%                      0-constant source/doublet
%                      1-constant doublet
%       err         -> Error
%       colldepth   -> Collocation point depth [m]
%       farfield    -> Far field coefficient
%       collcalc    -> Collocation point calculation:
%                      0-approximate
%                      1-accurate
%       velorder    -> Interpolation method/order for velocity calculations:
%                      0-nodal
%                      1-first
%                      2-second
%       results     -> Result requests:
%                      0-no
%                      1-yes
%       requests    -> Type of request: [vector 1x13] specify 1 or 0
%                      1-coefficients
%                      2-forces
%                      3-geometry
%                      4-velocity
%                      5-pressure
%                      6-center points
%                      7-dipole values
%                      8-source values
%                      9-velocity components
%                      10-mesh characteristics
%                      11-static pressure
%                      12-dynamic pressure
%                      13-manometer pressure
% Outputs:
%        Nopan -> Number of wing panels
function [NoPan,area] = ...
    BoxWingAPAME(Slices_Wing,h,name,airspeed,density,pressure,mach,cases,wingspan, ...
    MAC,surf,origin,method,err,colldepth,farfield,collcalc,velorder,results,requests)
addpath("sr_bwa\")

% Wing Panels
[Nx, Ny, Nz] = size(Slices_Wing);
for i = 1:Nz
    Slices_Wing(:,:,i) = Slices_Wing(linspace(Nx,1,Nx),:,i);
end
Slices_Wing(:,3,:) = Slices_Wing(:,3,:)-h/2;
aux_z = Nz;
for k = 1:Nz
    aux_z = aux_z + 1;
    for j = 1:Ny
        for i = 1:Nx
```

```
            if j == 2
                Slices_Wing(i,j,aux_z) = -Slices_Wing(i,j,Nz-k+1);
            else
                Slices_Wing(i,j,aux_z) = Slices_Wing(i,j,Nz-k+1);
            end
        end
    end
end
[Nx, Ny, Nz] = size(Slices_Wing);
aux_nodes = 0;
Wake_Points = 5;
Spacing_Wake = 10;
Nodes = (Nx-1)*Nz;
Nodes_List = zeros(Nodes,3);
Track_Nodes = zeros(Nx-1,Nz);
Track_Panels = zeros(Nx-1,Nz-2);
area = 0;
for k = 1:Nz
    for i = 1:Nx-1
        aux_nodes = aux_nodes+1;
        Nodes_List(aux_nodes,:) = Slices_Wing(i,:,k);
        Track_Nodes(i,k) = aux_nodes;
    end
end
aux_panels = 0;
for j = 1:Nz-2
    for i = 1:Nx-1
        aux_panels = aux_panels + 1;
        Track_Panels(i,j) = aux_panels;
    end
end
aux_pannodes = 0;
Panels = (Nz-2)*(Nx-1);
NoPan = Panels;
Panel_Nodes = zeros(Panels,4);
for j = 1:Nz-2
    for i = 1:Nx-1
        aux_pannodes = aux_pannodes + 1;
        if j < Nz/2
            if i == Nx-1
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(1,j);
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(1,j+1);
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+1);
            elseif i > Nx/2
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(i+1,j);
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(i+1,j+1);
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+1);
            else
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(i+1,j);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i+1,j+1);
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(i,j);
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+1);
            end
        elseif j == Nz-2
            if i == Nx-1
                Panel_Nodes(aux_pannodes,1) = Track_Nodes(1,j+1);
                Panel_Nodes(aux_pannodes,4) = Track_Nodes(1,end);
                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+1);
                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,end);
            elseif i > Nx/2
                Panel_Nodes(aux_pannodes,1) = Track_Nodes(i+1,j+1);
                Panel_Nodes(aux_pannodes,4) = Track_Nodes(i+1,end);
                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+1);
                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,end);
            else
                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i+1,j+1);
                Panel_Nodes(aux_pannodes,1) = Track_Nodes(i+1,end);
                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+1);
                Panel_Nodes(aux_pannodes,4) = Track_Nodes(i,end);
```

```
            end
        else
            if i == Nx-1
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(1,j+1);
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(1,j+2);
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+1);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+2);
            elseif i > Nx/2
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(i+1,j+1);
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(i+1,j+2);
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+1);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+2);
            else
               Panel_Nodes(aux_pannodes,2) = Track_Nodes(i+1,j+1);
               Panel_Nodes(aux_pannodes,1) = Track_Nodes(i+1,j+2);
               Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j+1);
               Panel_Nodes(aux_pannodes,4) = Track_Nodes(i,j+2);
            end
        end
        area_aux = quadrilateralArea(Nodes_List(Panel_Nodes(aux_pannodes,:),:));
        area = area + area_aux;
    end
end
aux_panconnect = 0;
Panel_Connectivity = zeros(Panels,4);
for j = 1:Nz-2
    for i = 1:Nx-1
        aux_panconnect = aux_panconnect + 1;
        if i == 1
            Panel_Connectivity(aux_panconnect,4) = 0;
            Panel_Connectivity(aux_panconnect,2) = Track_Panels(i+1,j);
            if j == 1
                Panel_Connectivity(aux_panconnect,1) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,end);
            elseif j == Nz-2
                Panel_Connectivity(aux_panconnect,1) = Track_Panels(i,1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,j-1);
            else
                Panel_Connectivity(aux_panconnect,1) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,j-1);
            end
        elseif i == Nx-1
            Panel_Connectivity(aux_panconnect,1) = Track_Panels(i-1,j);
            Panel_Connectivity(aux_panconnect,4) = 0;
            if j == 1
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,end);
            elseif j == Nz-2
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,j-1);
            else
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,3) = Track_Panels(i,j-1);
            end
        else
            Panel_Connectivity(aux_panconnect,1) = Track_Panels(i-1,j);
            Panel_Connectivity(aux_panconnect,3) = Track_Panels(i+1,j);
            if j == 1
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,4) = Track_Panels(i,end);
            elseif j == Nz-2
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,1);
                Panel_Connectivity(aux_panconnect,4) = Track_Panels(i,j-1);
            else
                Panel_Connectivity(aux_panconnect,2) = Track_Panels(i,j+1);
                Panel_Connectivity(aux_panconnect,4) = Track_Panels(i,j-1);
            end
        end
    end
end
Panel_List = ones(Panels,9);
```

B

```
208
209 % Wake panels
210 Panels = Panels + Wake_Points*(Nz-2);
211 Nodes  = Nodes + Wake_Points*Nz;
212
213 for k = 1:Nz
214     for i = 1:Wake_Points
215         aux_nodes = aux_nodes+1;
216         v_aux1 = Slices_Wing(1,:,k)-Slices_Wing(2,:,k); v_aux1 = v_aux1/norm(v_aux1);
217         v_aux2 = Slices_Wing(end,:,k)-Slices_Wing(end-1,:,k); v_aux2 = ...
                v_aux2/norm(v_aux2);
218         v_aux_avg = (v_aux1+v_aux2)/2; v_aux_avg = v_aux_avg/norm(v_aux_avg);
219         aux_wake = Spacing_Wake*i*v_aux_avg;
220         Nodes_List(aux_nodes,:) = aux_wake+Slices_Wing(1,:,k);
221         Track_Nodes(Nx-1+i,k) = aux_nodes;
222     end
223 end
224
225 for j = 1:Nz-2
226     for i = Nx-1+1:Nx-1+Wake_Points
227         aux_pannodes = aux_pannodes + 1;
228         if j < Nz/2
229             if i == Nx-1+1
230                Panel_Nodes(aux_pannodes,2) = Track_Nodes(1,j);
231                Panel_Nodes(aux_pannodes,1) = Track_Nodes(1,j+1);
232                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j);
233                Panel_Nodes(aux_pannodes,4) = Track_Nodes(i,j+1);
234             else
235                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i,j);
236                Panel_Nodes(aux_pannodes,4) = Track_Nodes(i,j+1);
237                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i-1,j);
238                Panel_Nodes(aux_pannodes,1) = Track_Nodes(i-1,j+1);
239             end
240         elseif j == Nz-2
241             if i == Nx-1+1
242                 Panel_Nodes(aux_pannodes,4) = Track_Nodes(1,j+1);
243                 Panel_Nodes(aux_pannodes,3) = Track_Nodes(1,end);
244                 Panel_Nodes(aux_pannodes,1) = Track_Nodes(i,j+1);
245                 Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,end);
246             else
247                 Panel_Nodes(aux_pannodes,1) = Track_Nodes(i,j+1);
248                 Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,end);
249                 Panel_Nodes(aux_pannodes,4) = Track_Nodes(i-1,j+1);
250                 Panel_Nodes(aux_pannodes,3) = Track_Nodes(i-1,end);
251             end
252         else
253             if i == Nx-1+1
254                Panel_Nodes(aux_pannodes,4) = Track_Nodes(1,j+1);
255                Panel_Nodes(aux_pannodes,3) = Track_Nodes(1,j+2);
256                Panel_Nodes(aux_pannodes,1) = Track_Nodes(i,j+1);
257                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+2);
258             else
259                Panel_Nodes(aux_pannodes,1) = Track_Nodes(i,j+1);
260                Panel_Nodes(aux_pannodes,2) = Track_Nodes(i,j+2);
261                Panel_Nodes(aux_pannodes,4) = Track_Nodes(i-1,j+1);
262                Panel_Nodes(aux_pannodes,3) = Track_Nodes(i-1,j+2);
263             end
264         end
265     end
266 end
267 for j = 1:Nz-2
268     for i = Nx-1+1:Nx-1+Wake_Points
269         aux_panconnect = aux_panconnect + 1;
270         Panel_Connectivity(aux_panconnect,1) = Track_Panels(1,j);
271         Panel_Connectivity(aux_panconnect,2) = Track_Panels(Nx-1,j);
272         Panel_Connectivity(aux_panconnect,3) = NaN;
273         Panel_Connectivity(aux_panconnect,4) = NaN;
274     end
275 end
276
277 for i = 1:Panels
```

```
278     Panel_List(i,2:5) = Panel_Nodes(i,:);
279     Panel_List(i,6:9) = Panel_Connectivity(i,:);
280     if Panel_List(i,1) == 0
281         Panel_List(i,1) = 10;
282     end
283 end
284 commandAPAME = ...
        inputFileAPAME(airspeed,density,pressure,mach,cases,wingspan,MAC,surf,origin, ...
        method,err,colldepth,farfield,collcalc,velorder,results,requests,Nodes, ...
        Nodes_List,Panels,Panel_List);
285
286 % Writing in file
287 fileName = fopen(name,'w');
288 fprintf(fileName,commandAPAME);
289 fclose(fileName);
290 end
```

B.2.1. APAME INPUT FILE

```
1  % This function generates an APAME input file command given the geometry
2  %       Author  : Gabriel Buendia
3  %       Version : 1
4  % Inputs:
5  %         airspeed     -> Freestream reference speed [m/s]
6  %         density      -> Freestream density [kg/m^3]
7  %         pressure     -> Freestream pressure [Pa]
8  %         mach         -> Fresstream Mach number
9  %         cases        -> Number of cases to simulate in APAME
10 %         wingspan     -> Wingspan [m]
11 %         MAC          -> Mean aerodynamic chord [m]
12 %         surf         -> Surface area [m]
13 %         origin       -> Origin of coordinates [m]
14 %         method       -> Singularity method:
15 %                         0-constant source/doublet
16 %                         1-constant doublet
17 %         err          -> Error
18 %         colldepth    -> Collocation point depth [m]
19 %         farfield     -> Far field coefficient
20 %         collcalc     -> Collocation point calculation:
21 %                         0-approximate
22 %                         1-accurate
23 %         velorder     -> Interpolation method/order for velocity calculations:
24 %                         0-nodal
25 %                         1-first
26 %                         2-second
27 %         results      -> Result requests:
28 %                         0-no
29 %                         1-yes
30 %         requests     -> Type of request: [vector 1x13] specify 1 or 0
31 %                         1-coefficients
32 %                         2-forces
33 %                         3-geometry
34 %                         4-velocity
35 %                         5-pressure
36 %                         6-center points
37 %                         7-dipole values
38 %                         8-source values
39 %                         9-velocity components
40 %                         10-mesh characteristics
41 %                         11-static pre> ->s>sure
42 %                         12-dynamic pressure
43 %                         13-manometer pressure
44 %         numbernodes  -> Number of nodes that the geometry contains
45 %         nodes        -> Geometry and wake nodes matrix [Nodes x 3] specifying:
46 %                         [x_pos y_pos z_pos] x Nodes
47 %         numberpanels -> Number of panels that the geometry contains
48 %         panels       -> Geometry and wake panels matrix [Panels x 7] specifying:
```

B

```
%                           [Type Node1 Node2 Node3 Node4 Panel1 Panel2 Panel3 Panel4] x ...
    Panels
%                           Type: 1  -> Square | 10  -> Wake
%                           Node1234: Nodes enclosing the panel
%                           Panel1234: Panels surrounding the actual panel
% Outputs:
%           commandAPAME -> String containing the command for an APAME input file
%                     to be written
function commandAPAME = ...
    inputFileAPAME(airspeed,density,pressure,mach,cases,wingspan,MAC,surf,origin, ...
    method,err,colldepth,farfield,collcalc,velorder,results,requests,numbernodes, ...
    nodes,numberpanels,panels)
[~,l_case] = size(cases); l_req = length(requests);
[r_nod, c_nod] = size(nodes); [r_pan, c_pan] = size(panels);
x0 = origin(1);
y0 = origin(2);
z0 = origin(3);
string = append('CASE_NUM ',num2str(l_case),'\n');
string1 = '';
string2 = '';
for i = 1:l_case
    if i < l_case
        string1 = append(string1,num2str(cases(1,i)),' ');
        string2 = append(string2,num2str(cases(2,i)),' ');
    else
        string1 = append(string1,num2str(cases(1,i)),'\n');
        string2 = append(string2,num2str(cases(2,i)),'\n');
    end
end
cases = append(string,string1,string2);
string3 = '';
for i = 1:r_nod
    for j = 1:c_nod
        if j < c_nod
            string3 = append(string3,num2str(nodes(i,j)),' ');
        else
            string3 = append(string3,num2str(nodes(i,j)),'\n');
        end
    end
end
nodes = string3;
string5 = '';
for i = 1:r_pan
    for j = 1:c_pan
        if j < c_pan
            if isnan(panels(i,j))
                string5 = append(string5,' ');
            else
                string5 = append(string5,num2str(panels(i,j)),' ');
            end
        else
            if isnan(panels(i,j))
                string5 = append(string5,'\n');
            else
                string5 = append(string5,num2str(panels(i,j)),'\n');
            end
        end
    end
end
panels = string5;
commandAPAME = append(['APAME input file\n'...
    'VERSION 3.1\n'...
    '# FLOW PARAMETERS\n'...
    '# airspeed [m/s]\n'...
    'AIRSPEED ',num2str(airspeed),'\n' ...
    '# air density [kg/m^3]\n' ...
    'DENSITY ',num2str(density),'\n' ...
    '# atmospheric pressure [Pa]\n' ...
    'PRESSURE ',num2str(pressure),'\n' ...
    '# prandtl-glauert correction:\n' ...
    '# 0-no correction\n' ...
```

```
'# *-Mach number\n' ...
'MACH ',num2str(mach),'\n' ...
'# number of cases\n' ...
'# angles of attack [degrees]\n' ...
'# sideslip angles [degrees]\n' ...
cases, ...
'# REFERENCE VALUES\n' ...
'# wing span [m]\n' ...
'WINGSPAN ',num2str(wingspan),'\n' ...
'# mean aerodynamic chord [m]\n' ...
'MAC ',num2str(MAC),'\n' ...
'# wing surface [m^2]\n' ...
'SURFACE ',num2str(surf),'\n' ...
'# reference point [m]\n' ...
'ORIGIN *\n' ...
num2str(x0),' ',num2str(y0),' ',num2str(z0),'\n' ...
'# SOLVER PARAMETERS\n' ...
'# singularity method:\n' ...
'# 0-constant source/doublet\n' ...
'# 1-constant doublet\n' ...
'METHOD ',num2str(method),'\n' ...
'# error\n' ...
'ERROR ',num2str(err),'\n' ...
'# collocation point depth\n' ...
'COLLDIST ',num2str(colldepth),'\n' ...
'# "far field" coefficient\n' ...
'FARFIELD ',num2str(farfield),'\n'...
'# collocation point calculation:\n' ...
'# 0-approximate\n' ...
'# 1-accurate\n' ...
'COLLCALC ',num2str(collcalc),'\n' ...
'# interpolation method/order for velocity calculations:\n' ...
'# 0-nodal\n' ...
'# 1-first\n' ...
'# 2-second\n' ...
'VELORDER ',num2str(velorder),'\n' ...
'# RESULT REQUESTS\n' ...
'# 0-no\n' ...
'# 1-yes\n' ...
'# RESULTS <1> or <0> (yes or no will the result file be written)\n' ...
'RESULTS ',num2str(results),'\n' ...
'#  1 coefficients\n' ...
'RES_COEF ',num2str(requests(1)),'\n' ...
'#  2 forces\n' ...
'RES_FORC ',num2str(requests(2)),'\n' ...
'#  3 geometry\n' ...
'RES_GEOM ',num2str(requests(3)),'\n' ...
'#  4 velocity\n' ...
'RES_VELO ',num2str(requests(4)),'\n' ...
'#  5 pressure\n' ...
'RES_PRES ',num2str(requests(5)),'\n' ...
'#  6 center points\n' ...
'RES_CENT ',num2str(requests(6)),'\n' ...
'#  7 dipole values\n' ...
'RES_DOUB ',num2str(requests(7)),'\n' ...
'#  8 source values\n' ...
'RES_SORC ',num2str(requests(8)),'\n' ...
'#  9 velocity components\n' ...
'RES_VELC ',num2str(requests(9)),'\n' ...
'# 10 mesh characteristics\n' ...
'RES_MESH ',num2str(requests(10)),'\n' ...
'# 11 static pressure\n' ...
'RES_STAT ',num2str(requests(11)),'\n' ...
'# 12 dynamic pressure\n' ...
'RES_DYNA ',num2str(requests(12)),'\n' ...
'# 13 manometer pressure\n' ...
'RES_MANO ',num2str(requests(13)),'\n' ...
'# 1 2 3 4 5 6 7 8 9 10 11 12 13\n' ...
'# GEOMETRY\n' ...
'# x y z [m]\n' ...
'NODES ',num2str(numbernodes),'\n' ...
```

```
    nodes, ...
    '# type node_id1 node_id2 node_id3 [node_id4] elem_id1 [elem_id2 [elem_id3 ...
        [elem_id4]]]\n' ...
    'PANELS ',num2str(numberpanels),'\n' ...
    panels, ...
    '# end of input file\n']);
```

B.2.2. QUADRILATERAL AREA

```
% This function computes the area of a quadrilateral element given 4
% points
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       points -> Vector [4x3] containing the coordinates [m] of the four
%                 points defining the quadrilateral element
% Outputs:
%       area -> Area of the quadrilateral element [m^2]
function area = quadrilateralArea(points)
vector = zeros(4,3);
vector(1,:) = points(2,:)-points(1,:);
vector(2,:) = points(4,:)-points(1,:);
vector(3,:) = points(4,:)-points(3,:);
vector(4,:) = points(2,:)-points(3,:);
cross_vec(1,:) = cross(vector(1,:),vector(2,:));
cross_vec(2,:) = cross(vector(3,:),vector(4,:));
area_1 = 0.5*norm(cross_vec(1,:));
area_2 = 0.5*norm(cross_vec(2,:));
area = area_1 + area_2;
end
```

B.3. BOX WING POINTWISE

```
% This function generates the glyph file to be run by Pointwise to generate
% the CFD mesh
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       name          -> Name of the input file
%       Slices_Wing   -> Matrix [N_points_airfoil x 3 x N_slices] containing
%                          all the wing cross sections [m]
%       b             -> Wingspan [m]
%       R             -> Cant radius [m]
%       h             -> Height between wings [m]
%       s             -> Wing stagger [m]
%       c_r1          -> Chord at the root of the lower wing [m]
%       c_t1          -> Chord at the tip of the lower wing [m]
%       c_r2          -> Chord at the root of the upper wing [m]
%       c_t2          -> Chord at the tip of the upper wing [m]
%       N             -> Number of sections to discretize the cant region
%       N_bu          -> Number of spanwise sections upper wing
%       N_bl          -> Number of spanwise sections lower wing
%       N_bw          -> Number of spanwise sections winglet
%       el_size       -> Size of the element [m]
%       tolBound      -> This attribute is the tolerance to use when splitting
%                        and joining the curves specified as the boundaries, for
%                        detecting end to end connections of the boundaries, and
%                        for creating the surface from the boundaries
%       tolThres      -> This attribute specifies the percentage of length of
%                        the boundary curves of a created surface that must be
%                        database constrained to automatically set the fitting
%                        entities
%       AR            -> Aspect ratio of the desired cells
%       nodesTEz      -> Nodes to discretize finite trailing edge
%       advanced      -> Structure containing options for unstructured mesh only
```

```
%                     -Fields:
%                     --IsoCellType              -> 'Triangle'||'TriangleQuad'
%                     --Algorithm                -> 'Delaunay (Default)'||
%                                                   'AdvancingFront'||
%                                                   'AdvancingFrontOrtho'||
%                                                   'ThinSurfaceInterpolation'
%                     --EdgeMinimumLength        -> Number||
%                                                   'Automatic (Default)'||
%                                                   'Boundary'||
%                                                   'TRexBoundary'||
%                                                   'NotApplied'
%                     --EdgeMaximumLength        -> Number||
%                                                   'Automatic (Default)'||
%                                                   'Boundary'||
%                                                   'TRexBoundary'||
%                                                   'NotApplied'
%                     --NormalMaximumDeviation   -> Number (Default 0)
%                     --NormalSurfaceDeviation   -> Number (Default 0)
%                     --QuadMaximumIncludedAngle -> Number (Default 150)
%                     --QuadMaximumWarpAngle     -> Number (Default 30)
%       FF          -> Structure containing options for defining the farfield
%                     -Fields:
%                     --Type -> 'Box' : Box shape farfield
%                               'Cylinder' : Cylindrical shape farfield
%                               'Sphere' : Spherical shape farfield
%                     --Cell -> 'Unstructured' : Unstructured elements in the FF
%                               'Voxel' : Voxel elements in the FF
%                     --Dims -> They depend on the 'optionsFF' specified
%                               if 'Box'      -> 3x2 Matrix : [x_positive x_negative;
%                                                              y_positive y_negative;
%                                                              z_positive z_negative]
%                               if 'Cylinder' -> 3x2 Matrix : [L_positive L_negative;
%                                                              Radius_1   Radius_2;
%                                                              Axis_alignment    0]
%                               if 'Sphere'   -> 1x1 Variable : Radius
%      BLoptions    -> Structure containing options for the boundary layer
%                     -Fields:
%                     --maxLayers        -> This default is the maximum number of
%                     T-Rex layers of an unstructured block when it is created :
%                     Number (Default 0)
%                     --fullLayers       -> This default is the minimum number of
%                     fully structured T-Rex layers of an unstructured block when
%                     it is created : Number (Default 0)
%                     --growthRate       -> This default is the growth rate of
%                     T-Rex layers of an unstructured block when it is created :
%                     Number (Default 1.2)
%                     --push             -> This default is the flag for pushing
%                     T-Rex attributes onto the connectors and domains of an
%                     unstructured block when a it is created : 'true' or 'false'
%                     (Default false)
%                     --solverAttribute  -> Sets the named unstructured solver
%                     attribute : 'TetPyramidPrismHex'||'AllAndReducePyramids'||
%                                 'AllAndConvertWallDoms'||'LegacyTetPyramidPrismHex'||
%                                 'TetPyramid'
%                     --firstLayerHeight  -> Specifies the BL thickness of the
%                     first cell : Number (User specified)
%      ISOoptions   -> -minLength -> Minimum length of the isotropic tets :
%                      Number||'Boundary' (Default Boundary)
%                      -maxLength -> Maximum length of the isotropic tets :
%                      Number||'Boundary' (Default Boundary)
%      sources      -> Cell array {1xN_source}
%                     Each of the elements of the cell array is a structure
%                     defining the source.
%                     -Fields:
%                     --Type        -> 'Box' : Box shape source
%                                      'Cylinder' : Cylindrical shape source
%                                      'Sphere' : Spherical shape source
%                     --Startface   -> 'x' : Source start and end faces are defined
%                                      following the unitary vector i (x axis)
%                                      'y' : Source start and end faces are defined
%                                      following the unitary vector j (y axis)
```

```
%                                       'z' : Source start and end faces are defined
%                                      following the unitary vector k (z axis)
%                       --Center      -> Specify the COG of the source in cartesian coords
%                                        1x3 Vector : [x_center y_center z_center]
%                        --Dims        -> They depend on the 'optionsSrc' specified
%                                        if 'Box'      -> 1x3 Vector :
%                                                         [length height width]
%                                        if 'Cylinder' -> 1x3 Vector :
%                                                         [radius top_Radius length]
%                                                         Note that radius and top_Radius
%                                                         are the same for cylinder and
%                                                         different if defining a cone
%                                        if 'Sphere'   -> 1x3 Vector :
%                                                         [radius base_Angle top_Angle]
%                                                         Note that base_Angle ->
%                                                         [0,90] deg and top_Angle ->
%                                                         [90,180] deg and are used to
%                                                         define cut sphere sources
%                       --Distr       -> 'Constant': The begin values will be used
%                                      throughout the source
%                                       'Parametric' : The begin values will be used at
%                                      the minimum parametric limits and the end values
%                                      will be used at the maximum parametric limits of
%                                      the source
%                                       'AxisToPerimeter : The begin values will be
%                                      used along the axis and the end values will be
%                                      used at the perimeter of the source
%                                       'CenterToPerimeter : The begin values will be
%                                      used at the center and the end values will be
%                                      used at the perimeter of the source
%                      --Distrval     -> 2x2 Matrix: [spacing_initial spacing_final;
%                                                     decay_initial   decay_final]
%                                      Note that for the 'constant' case only the
%                                      initial values are read
%      surfDom       -> Structure containing specification for surface mesh in terms of
%                       'structured' or 'unstructured'.
%                       Fields:
%                       -wings        -> Specify the structure on the upper and lower
%                                        wings
%                       -cant         -> Specify the structure on the cant regions
%                       -winglets     -> Specify the structure on the winglets
%       distribution -> Connector grid point distribution function is
%                       specified : 'MRQS' | 'Tanh'
%       refineVal    -> Minimum node spacing in the connector
%       N_cant       -> Number of nodes in the cant region
%
%       fixedPoint   -> Vector (1x3) defining the fixed point for the rotation
%                       [xRot yRot zRot]
%       rotAxis      -> if 'x' : The rotation axis is x
%                        if 'y' : The rotation axis is y
%                        if 'z' : The rotation axis is z
%       rotAngle     -> Angle to be rotated [deg]
%       exportPath   -> String that specifies where the mesh is exported to
%       savePath     -> String that specifies the path for the file to be saved
%       saveName     -> String that specifies the name of the file to be saved
function BoxWingGlyph(name,Slices_Wing,b,R,h,s,c_r1,c_t1,c_r2,c_t2,N,N_bl,N_bu,N_bw, ...
    el_size,tolBound,tolThres,AR,nodesTEz,advanced,FF,symmetry,BLoptions,ISOoptions, ...
    sources,surfDom,distribution,refineVal,N_cant,fixedPoint,rotAxis,rotAngle, ...
    exportPath,savePath,saveName)
addpath("sr_bwg\sr_aux\")
addpath("sr_bwg\sr_pw\")
% File
fileName = fopen(name,'w');
fprintf(fileName,'package require PWI_Glyph 5.18.5\n');

% Lines and sections
[N_lines,~,N_slices] = size(Slices_Wing);

% IDs
point_ID = zeros(N_slices,N_lines-1);
line_ID = zeros(N_slices,6);
```

```
surface_ID = zeros(N_slices-1,3);

% Counter Limits
counter_points = 0;
LE = N_lines/2;
counter_lines = 10^(ceil(log10(N_slices))*ceil(log10(N_lines-1)));
counter_surfaces = 10^(ceil(log10(N_slices))*ceil(log10(N_lines-1))+1);
counter_farfield = 10^(ceil(log10(N_slices))*ceil(log10(N_lines-1))+2);
counter_source = 10^(ceil(log10(N_slices))*ceil(log10(N_lines-1))+3);

% Surface Mesh Options
[N_LEls, N_LEus, N_AF, N_R, N_WL] = ...
    computeNumberOfElements(el_size,b,R,h,s,c_r1,c_t1,c_r2,c_t2,N,N_bl,N_bu,N_bw,AR);
nodesAirfoil = N_AF;

% Farfield
if isequal(symmetry,'off')
    dim = [s b h];
    arrangement = 2*ones(1,3);
    [~,posMax] = max(dim); arrangement(posMax) = 1;
    [~,posMin] = min(dim); arrangement(posMin) = 3;
else
    dim = [s b/2 h];
    arrangement = 2*ones(1,3);
    [~,posMax] = max(dim); arrangement(posMax) = 1;
    [~,posMin] = min(dim); arrangement(posMin) = 3;
end

% Sources
sources_size = length(sources);

%% Computations
for i = 1:N_slices
    for j = 1:N_lines-1
        counter_points = counter_points+1;
        point_ID(i,j) = counter_points;
        pointCommand = pwPoints(counter_points,Slices_Wing(j,1,i), ...
            Slices_Wing(j,2,i),Slices_Wing(j,3,i));
        fprintf(fileName,pointCommand);
    end
    if (i < N_bl+3)||(i > 2*N+N_bl+N_bw+1) %((i > 16)&&(i < 21))
            domainType = surfDom.wings;
            nodesLE = N_LEls;
            nodesTE1y = N_LEls;
            nodesTE2y = N_LEls;
    elseif ((i > N+N_bl+1)&&(i < N+N_bl+N_bw+3))
            domainType = surfDom.winglet;
            nodesLE = N_WL;
            nodesTE1y = N_WL;
            nodesTE2y = N_WL;
    else
            domainType = surfDom.cant;
            nodesLE = N_cant;
            nodesTE1y = N_cant;
            nodesTE2y = N_cant;
    end
    % Exterior curve
    counter_lines = counter_lines+1;
    line_ID(i,1) = counter_lines;
    curveCommand = pwCurve(counter_lines,point_ID(i,1:LE));
    connectorCommand = ...
        pwConnector(line_ID(i,1),nodesAirfoil,line_ID(i,1),'dimension',distribution, ...
        refineVal);
    fprintf(fileName,curveCommand);
    fprintf(fileName,connectorCommand);
    % Interior curve
    counter_lines = counter_lines+1;
    line_ID(i,2) = counter_lines;
    curveCommand = pwCurve(counter_lines,point_ID(i,LE:end));
    connectorCommand = ...
        pwConnector(line_ID(i,2),nodesAirfoil,line_ID(i,2),'dimension',distribution, ...
```

```
            refineVal);
238     fprintf(fileName,curveCommand);
239     fprintf(fileName,connectorCommand);
240     % TE in z direction
241     counter_lines = counter_lines+1;
242     line_ID(i,3) = counter_lines;
243     curveCommand = pwCurve(counter_lines,[point_ID(i,end) point_ID(i,1)]);
244     connectorCommand = pwConnector(line_ID(i,3),nodesTEz,line_ID(i,3),'dimension');
245     fprintf(fileName,curveCommand);
246     fprintf(fileName,connectorCommand);
247
248     if i ≤ 1
249         line_ID(i,4:6) = NaN;
250     else
251         % LE
252         counter_lines = counter_lines+1;
253         line_ID(i,4) = counter_lines;
254         curveCommand = pwCurve(counter_lines,[point_ID(i-1,LE) point_ID(i,LE)]);
255         connectorCommand = pwConnector(line_ID(i,4),nodesLE,line_ID(i,4),'dimension');
256         fprintf(fileName,curveCommand);
257         fprintf(fileName,connectorCommand);
258         % TE1 in y direction (lower)
259         counter_lines = counter_lines+1;
260         line_ID(i,5) = counter_lines;
261         curveCommand = pwCurve(counter_lines,[point_ID(i-1,1) point_ID(i,1)]);
262         connectorCommand = pwConnector(line_ID(i,5),nodesTE1y,line_ID(i,5),'dimension');
263         fprintf(fileName,curveCommand);
264         fprintf(fileName,connectorCommand);
265         % TE2 in y direction (upper)
266         counter_lines = counter_lines+1;
267         line_ID(i,6) = counter_lines;
268         curveCommand = pwCurve(counter_lines,[point_ID(i-1,end) point_ID(i,end)]);
269         connectorCommand = pwConnector(line_ID(i,6),nodesTE2y,line_ID(i,6),'dimension');
270         fprintf(fileName,curveCommand);
271         fprintf(fileName,connectorCommand);
272         %Surfaces
273         % Exterior surface
274         counter_surfaces = counter_surfaces+1;
275         surface_ID(i-1,1) = counter_surfaces;
276         surfaceCommand = pwSurface(counter_surfaces,[line_ID(i-1,1) line_ID(i,4) ...
                line_ID(i,1) line_ID(i,5)],tolBound,tolThres);
277         domainCommand = pwDomain(counter_surfaces,[line_ID(i-1,1) line_ID(i,4) ...
                line_ID(i,1) line_ID(i,5)],domainType,advanced);
278         fprintf(fileName,surfaceCommand);
279         fprintf(fileName,domainCommand);
280         % Interior surface
281         counter_surfaces = counter_surfaces+1;
282         surfaceCommand = pwSurface(counter_surfaces,[line_ID(i-1,2) line_ID(i,4) ...
                line_ID(i,2) line_ID(i,6)],tolBound,tolThres);
283         domainCommand = pwDomain(counter_surfaces,[line_ID(i-1,2) line_ID(i,4) ...
                line_ID(i,2) line_ID(i,6)],domainType,advanced);
284         fprintf(fileName,surfaceCommand);
285         fprintf(fileName,domainCommand);
286         surface_ID(i-1,2) = counter_surfaces;
287         % TE surface
288         domainType = 'structured';
289         counter_surfaces = counter_surfaces+1;
290         surfaceCommand = pwSurface(counter_surfaces,[line_ID(i-1,3) line_ID(i,6) ...
                line_ID(i,3) line_ID(i,5)],tolBound,tolThres);
291         domainCommand = pwDomain(counter_surfaces,[line_ID(i-1,3) line_ID(i,6) ...
                line_ID(i,3) line_ID(i,5)],domainType);
292         fprintf(fileName,surfaceCommand);
293         fprintf(fileName,domainCommand);
294         surface_ID(i-1,3) = counter_surfaces;
295     end
296     counter_points = counterUpdate(N_lines-1,counter_points);
297     counter_lines = counterUpdate(6,counter_lines);
298     counter_surfaces = counterUpdate(3,counter_surfaces);
299 end
300
301 % Angle of attack rotation
```

```
rotationCommand = pwRotation(fixedPoint,rotAxis,rotAngle,'allDataBaseandGrid',NaN, ...
    [point_ID(:);line_ID(:);surface_ID(:)],line_ID(:),surface_ID(:));
fprintf(fileName,rotationCommand);

% Symmetry copy
if isequal(symmetry,'off')
    mirrorCommand = ...
        pwMirror([point_ID(:);line_ID(:);surface_ID(:)],line_ID(:),surface_ID(:),'y');
    fprintf(fileName,mirrorCommand);
    domainNumber = 2*length(surface_ID(:));
else
    domainNumber = length(surface_ID(:));
end

% Source in between wings
sourceCommand = pwSource(counter_source,sources{1}.Type,sources{1}.StartFace, ...
    sources{1}.Center,sources{1}.Dims,sources{1}.Distr,sources{1}.DistrVal);
fprintf(fileName,sourceCommand);
rotationCommand = pwRotation(fixedPoint,rotAxis,rotAngle,'allSource',NaN,NaN,NaN,NaN);
fprintf(fileName,rotationCommand);

% Farfield generation
farfieldCommand = pwFarfield(counter_farfield,0,FF.Type,FF.TypeCell,FF.Dims,arrangement);
fprintf(fileName,farfieldCommand);

% Rest of sources (refinements)
if sources_size > 1
    for i = 2:sources_size
        counter_source = counter_source+1;
        sourceCommand = pwSource(counter_source,sources{i}.Type,sources{i}.StartFace, ...
            sources{i}.Center,sources{i}.Dims,sources{i}.Distr,sources{i}.DistrVal);
        fprintf(fileName,sourceCommand);
        rotationCommand = ...
            pwRotation(fixedPoint,rotAxis,rotAngle/2,'otherSource',counter_source, ...
            NaN,NaN,NaN);
        fprintf(fileName,rotationCommand);
    end
end

% Volumetric mesh generation
volumeMeshCommand = pwMesh(counter_farfield, BLoptions, ISOoptions);
fprintf(fileName,volumeMeshCommand);

% Export and save commands
exportCommand = pwExportOPF(counter_farfield,domainNumber,'3D',exportPath);
fprintf(fileName,exportCommand);
saveandfinishCommand = pwSaveAndFinish(savePath,saveName);
fprintf(fileName,saveandfinishCommand);
fclose(fileName);
```

B.3.1. COMPUTE NUMBER OF ELEMENTS

```
% This function computes the number of elements given an element size, some
% geometrical parameters and the aspect ratio of the desired cells
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       el_size  -> Size of the element [m]
%       b        -> Wingspan [m]
%       R        -> Cant radius [m]
%       h        -> Height between wings [m]
%       s        -> Wing stagger [m]
%       c_r1     -> Chord at the root of the lower wing [m]
%       c_t1     -> Chord at the tip of the lower wing [m]
%       c_r2     -> Chord at the root of the upper wing [m]
%       c_t2     -> Chord at the tip of the upper wing [m]
%       N        -> Number of sections to discretize the cant region
%       N_bu     -> Number of spanwise sections upper wing
```

```
% N_bl     -> Number of spanwise sections lower wing
% N_bw     -> Number of spanwise sections winglet
% AR       -> Aspect ratio of the desired cells
% Outputs:
% N_LEls   -> Number of elements in the lower wing leading edge
% N_LEus   -> Number of elements in the upper wing leading edge
% N_AF     -> Number of elements present in the airfoil
% N_R      -> Number of elements in the leading edge of the cant region
% N_WL     -> Number of elements in the winglet
function [N_LEls, N_LEus, N_AF, N_R, N_WL] = ...
    computeNumberOfElements(el_size,b,R,h,s,c_r1,c_t1,c_r2,c_t2,N,N_bl,N_bu,N_bw,AR)
c_min = min([c_r1,c_r2,c_t1,c_t2]);
N_LEls = 1+ceil((b/2-R)/((N_bl+1)*el_size)/AR);
N_LEus = 1+ceil((b/2-R)/((N_bu+1)*el_size)/AR);
N_AF = 1+ceil(c_min/el_size);
N_R = 1+ceil(pi*R/(2*(N-1))/el_size/AR);
N_WL = 1+ceil((sqrt(h^2+s^2)-2*R)/((N_bw+1)*el_size)/AR);
end
```

B.3.2. COUNTER UPDATE

```
% This function aim to reset the counter for each airfoil section
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       N       -> Number of points in a section
%       counter -> Actual value of the counter at the end of the section
% Output:
%       counter -> Updated counter to the next decimal place
function counter = counterUpdate(N,counter)
digits = ceil(log10(N));
counter = counter/10^digits;
counter = ceil(counter);
counter = counter*10^digits;
end
```

B.3.3. POINTWISE CONNECTOR

```
% This function aims to generate a connector with nodes on a curve (1D mesh)
% for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID           -> Pointwise ID desired connector number
%       dim          -> Expected nodes on the connector or spacing
%       ID_DB        -> Pointwise ID of the database curve
%       spacing      -> 'dimension' : Nodes are created specifying a dimension
%                       'separation' : Nodes are created specifying an average spacing
%       distribution -> Connector grid point distribution function is
%                       specified : 'MRQS' | 'Tanh'
%       refineVal    -> Minimum node spacing in the connector
% Outputs:
%       connectorCommand -> String containing the command to write the
%                           connector
function connectorCommand = pwConnector(ID,dim,ID_DB,spacing,distribution,refineVal)
switch spacing
    case 'dimension'
    dimension = append('pw::Connector setDefault Dimension ',num2str(dim),'\n  ');
    case 'separation'
    dimension = append(['pw::Connector setCalculateDimensionMethod Spacing\n' ...
        'pw::Connector setCalculateDimensionSpacing '],num2str(dim),'\n  ');
end

connectorCommand = append([dimension, ...
    'set _CN(',num2str(ID),') [pw::Connector createOnDatabase ...
        $_DB(',num2str(ID_DB),')]\n', ...
```

```
    '$_CN(',num2str(ID),') replaceDistribution 1 [pw::DistributionTanh create]\n']);
if nargin > 4
    distributionCommand = append(['set _TMP(mode_1) [pw::Application begin Modify ...
        [list $_CN(',num2str(ID),')]]\n', ...
                                 '  pw::Connector swapDistribution ',distribution,' ...
                                    [list [list $_CN(',num2str(ID),') 1]]\n', ...
                                 '  [[$_CN(',num2str(ID),') getDistribution 1] ...
                                    getEndSpacing] setValue ',num2str(refineVal),'\n', ...
                                 '  [[$_CN(',num2str(ID),') getDistribution 1] ...
                                    getBeginSpacing] setValue ...
                                    ',num2str(refineVal),'\n', ... '
                                '$_TMP(mode_1) end\n', ...
                                'unset _TMP(mode_1)\n']);
    connectorCommand = append(connectorCommand,distributionCommand);
end
end
```

B.3.4. POINTWISE CURVE

```
% This function aims to generate single curve from a list of points
% for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID      -> Pointwise ID desired database number
%       list    -> List of points IDs that belong to the curve
%       options -> NaN : Simple curve joining points
%                  1   : Spline using Catmull-Rom algorithm
%                  2   : Spline using Akima algorithm
% Outputs:
%       curveCommand -> String containing the command to write the curve
function curveCommand = pwCurve(ID,list,options)
size = length(list);
string = '';
for i = 1:size
    auxstr = strcat('  $_TMP(PW_1) addPoint [list 0 0 $_DB(', ...
        num2str(list(i)),')]\n');
    string = append(string,auxstr);
end
if nargin < 3
    options = '';
else
    switch options
        case 1
            options = '  $_TMP(PW_1) setSlope CatmullRom\n';
        case 2
            options = '  $_TMP(PW_1) setSlope Akima\n';
    end
end
curveCommand = append(['set _TMP(mode_1) [pw::Application begin Create]\n  ' ...
    'set _TMP(PW_1) [pw::SegmentSpline create]\n'], ...
    string, ...
    options, ...
    '  set _DB(',num2str(ID),[') [pw::Curve create]\n' ...
    '  $_DB(',num2str(ID),') addSegment $_TMP(PW_1)\n' ...
    '  unset _TMP(PW_1)\n$_TMP(mode_1) end\n' ...
    'unset _TMP(mode_1)\n']);
end
```

B.3.5. POINTWISE DOMAIN

```
% This function aims to generate a domain (2D mesh) from a list of
% connectors for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
```

B

```
%       ID       -> Pointwise ID desired domain number
%       list     -> List of connector IDs enclosing the domain
%       options  -> 'structured' : Structured domain
%                   'unstructured' : Unstructured domain
%       advanced -> Structure containing options for unstructured mesh only
%                   Fields:
%                   -IsoCellType              -> 'Triangle'||'TriangleQuad'
%                   -Algorithm                -> 'Delaunay (Default)'||
%                                                'AdvancingFront'||
%                                                'AdvancingFrontOrtho'||
%                                                'ThinSurfaceInterpolation'
%                   -EdgeMinimumLength        -> Number||
%                                                'Automatic (Default)'||
%                                                'Boundary'||
%                                                'TRexBoundary'||
%                                                'NotApplied'
%                   -EdgeMaximumLength        -> Number||
%                                                'Automatic (Default)'||
%                                                'Boundary'||
%                                                'TRexBoundary'||
%                                                'NotApplied'
%                   -NormalMaximumDeviation   -> Number (Default 0)
%                   -NormalSurfaceDeviation   -> Number (Default 0)
%                   -QuadMaximumIncludedAngle -> Number (Default 150)
%                   -QuadMaximumWarpAngle     -> Number (Default 30)
% Outputs:
%       domainCommand -> String containing the command to write the
%                        domain
function domainCommand = pwDomain(ID,list,options,advanced)
size = length(list);
string = '[list ';
for i = 1:size
    if i < size
        auxstr = append('$_CN(',num2str(list(i)),') ');
    else
        auxstr = append('$_CN(',num2str(list(i)),')]');
    end
    string = append(string,auxstr);
end
switch options
    case 'structured'
        domainCommand = append('set _DM(',num2str(ID),') [pw::DomainStructured ...
            createFromConnectors ',string,']\n');
    case 'unstructured'
        domainCommand = append('set _DM(',num2str(ID),') [pw::DomainUnstructured ...
            createFromConnectors ',string,']\n');
        if nargin > 3
            IsoCellType = advanced.IsoCellType;

            if sum(strcmp(fieldnames(advanced), 'Algorithm')) == 1
                Algorithm = advanced.Algorithm;
            else
                Algorithm = 'Delaunay';
            end

            if sum(strcmp(fieldnames(advanced), 'EdgeMinimumLength')) == 1
                EdgeMinimumLength = advanced.EdgeMinimumLength;
            else
                EdgeMinimumLength = 'Automatic';
            end

            if sum(strcmp(fieldnames(advanced), 'EdgeMaximumLength')) == 1
                EdgeMaximumLength = advanced.EdgeMaximumLength;
            else
                EdgeMaximumLength = 'Automatic';
            end

            if sum(strcmp(fieldnames(advanced), 'NormalMaximumDeviation')) == 1
                NormalMaximumDeviation = advanced.NormalMaximumDeviation;
            else
                NormalMaximumDeviation = 0;
```

```
75              end
76
77              if sum(strcmp(fieldnames(advanced), 'SurfaceMaximumDeviation')) == 1
78                  SurfaceMaximumDeviation = advanced.NormalMaximumDeviation;
79              else
80                  SurfaceMaximumDeviation = 0;
81              end
82
83              if isequal(IsoCellType,'TriangleQuad')
84                  if sum(strcmp(fieldnames(advanced), 'QuadMaximumWarpAngle')) == 1
85                      QuadMaximumIncludedAngle = advanced.QuadMaximumIncludedAngle;
86                  else
87                      QuadMaximumIncludedAngle = 150;
88                  end
89
90                  if sum(strcmp(fieldnames(advanced), 'QuadMaximumWarpAngle')) == 1
91                      QuadMaximumWarpAngle = advanced.QuadMaximumWarpAngle;
92                  else
93                      QuadMaximumWarpAngle = 30;
94                  end
95                  domainCommand = append([domainCommand,...
96                      'set _TMP(mode_1) [pw::Application begin UnstructuredSolver [list ...
                            $_DM(',num2str(ID),')]]\n'...
97                      '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            IsoCellType ',IsoCellType,'\n'...
98                      '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            Algorithm ',Algorithm,'\n'...
99                      '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            EdgeMinimumLength ',num2str(EdgeMinimumLength),'\n'...
100                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            EdgeMaximumLength ',num2str(EdgeMaximumLength),'\n'...
101                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            NormalMaximumDeviation ',num2str(NormalMaximumDeviation),'\n'...
102                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            SurfaceMaximumDeviation ...
                            ',num2str(SurfaceMaximumDeviation),'\n'...
103                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            QuadMaximumIncludedAngle ...
                            ',num2str(QuadMaximumIncludedAngle),'\n'...
104                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            QuadMaximumWarpAngle ',num2str(QuadMaximumWarpAngle),'\n'...
105                     '   $_TMP(mode_1) run Initialize\n' ...
106                     '$_TMP(mode_1) end\n']);
107             else
108                 domainCommand = append([domainCommand,...
109                     'set _TMP(mode_1) [pw::Application begin UnstructuredSolver [list ...
                            $_DM(',num2str(ID),')]]\n'...
110                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            IsoCellType ',IsoCellType,'\n'...
111                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            Algorithm ',Algorithm,'\n'...
112                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            EdgeMinimumLength ',num2str(EdgeMinimumLength),'\n'...
113                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            EdgeMaximumLength ',num2str(EdgeMaximumLength),'\n'...
114                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            NormalMaximumDeviation ',num2str(NormalMaximumDeviation),'\n'...
115                     '   $_DM(',num2str(ID),') setUnstructuredSolverAttribute ...
                            SurfaceMaximumDeviation ...
                            ',num2str(SurfaceMaximumDeviation),'\n'...
116                     '   $_TMP(mode_1) run Initialize\n' ...
117                     '$_TMP(mode_1) end\n']);
118             end
119         end
120 end
121 end
```

B.3.6. POINTWISE EXPORT TO OPENFOAM

B

```
% This function aims to import the volume mesh from Pointwise to openFOAM
% format. It is made by dafault for the case in which the wing is simulated
% within a box farfield without the symmetry option on
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%      ID            -> ID of the already existing bulk data entity
%      numberDomains -> Number of domains the aerodynamic profile is composed of
%      dimension     -> '2D' or '3D'
%      path          -> String that specifies where the mesh is exported to
% Outputs:
%      exportCommand -> String containing the command to export the mesh to
%                       openFOAM
function exportCommand = pwExportOPF(ID,numberDomains,dimension,path)
string = '$_TMP(PW_5) apply [list ';
for i = 1:numberDomains
    if i < numberDomains
        auxstring = append('[list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
            dom-',num2str(i),']] ');
    else
        auxstring = append('[list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
            dom-',num2str(i),']]]\n');
    end
    string = append(string,auxstring);
end
exportCommand = append(['pw::Application setCAESolver OpenFOAM 3\n' ...
    'pw::Application markUndoLevel {Set Dimension ',dimension,'}\n' ...
    'set _TMP(PW_1) [pw::BoundaryCondition create]\n' ...
    'pw::Application markUndoLevel {Create BC}\n' ...
    'unset _TMP(PW_1)\n' ...
    'set _TMP(PW_1) [pw::BoundaryCondition getByName bc-2]\n' ...
    '$_TMP(PW_1) setName inlet\n' ...
    '$_TMP(PW_1) setPhysicalType -usage CAE patch\n' ...
    'pw::Application markUndoLevel {Name BC}\n' ...
    '$_TMP(PW_1) apply [list [list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
        dom-',num2str(numberDomains+6),']]]\n' ...
    'pw::Application markUndoLevel {Set BC}\n' ...
    'set _TMP(PW_2) [pw::BoundaryCondition create]\n' ...
    'pw::Application markUndoLevel {Create BC}\n' ...
    'unset _TMP(PW_2)\n' ...
    'set _TMP(PW_2) [pw::BoundaryCondition getByName bc-3]\n' ...
    '$_TMP(PW_2) setName outlet\n' ...
    '$_TMP(PW_2) setPhysicalType -usage CAE patch\n' ...
    'pw::Application markUndoLevel {Name BC}\n' ...
    '$_TMP(PW_2) apply [list [list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
        dom-',num2str(numberDomains+4),']]]\n' ...
    'pw::Application markUndoLevel {Set BC}\n' ...
    'set _TMP(PW_3) [pw::BoundaryCondition create]\n' ...
    'pw::Application markUndoLevel {Create BC}\n' ...
    'unset _TMP(PW_3)\n' ...
    'set _TMP(PW_3) [pw::BoundaryCondition getByName bc-4]\n' ...
    '$_TMP(PW_3) setName upAndDown\n' ...
    '$_TMP(PW_3) setPhysicalType -usage CAE patch\n' ...
    'pw::Application markUndoLevel {Name BC}\n' ...
    '$_TMP(PW_3) apply [list [list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
        dom-',num2str(numberDomains+3),']] [list $_BL(',num2str(ID),') ...
        [pw::GridEntity getByName dom-',num2str(numberDomains+5),']]]\n' ...
    'pw::Application markUndoLevel {Set BC}\n' ...
    'set _TMP(PW_4) [pw::BoundaryCondition create]\n' ...
    'pw::Application markUndoLevel {Create BC}\n' ...
    'unset _TMP(PW_4)\n' ...
    'set _TMP(PW_4) [pw::BoundaryCondition getByName bc-5]\n' ...
    '$_TMP(PW_4) setName back\n' ...
    '$_TMP(PW_4) setPhysicalType -usage CAE patch\n' ...
    'pw::Application markUndoLevel {Name BC}\n' ...
    '$_TMP(PW_4) apply [list [list $_BL(',num2str(ID),') [pw::GridEntity getByName ...
        dom-',num2str(numberDomains+1),']] [list $_BL(',num2str(ID),') ...
        [pw::GridEntity getByName dom-',num2str(numberDomains+2),']]]\n' ...
    'pw::Application markUndoLevel {Set BC}\n' ...
    'set _TMP(PW_5) [pw::BoundaryCondition create]\n' ...
```

```
    'pw::Application markUndoLevel {Create BC}\n' ...
    'unset _TMP(PW_5)\n' ...
    'set _TMP(PW_5) [pw::BoundaryCondition getByName bc-6]\n' ...
    '$_TMP(PW_5) setName cylinder\n' ...
    '$_TMP(PW_5) setPhysicalType -usage CAE wall\n' ...
    'pw::Application markUndoLevel {Name BC}\n' ...
    string, ...
    'pw::Application markUndoLevel {Set BC}\n' ...
    'unset _TMP(PW_1)\n' ...
    'unset _TMP(PW_2)\n' ...
    'unset _TMP(PW_3)\n' ...
    'unset _TMP(PW_4)\n' ...
    'unset _TMP(PW_5)\n' ...
    'set _TMP(PW_1) [pw::VolumeCondition create]\n' ...
    'pw::Application markUndoLevel {Create VC}\n' ...
    '$_TMP(PW_1) setName Volume\n' ...
    'pw::Application markUndoLevel {Name VC}\n' ...
    '$_TMP(PW_1) apply [list $_BL(',num2str(ID),')]\n' ...
    'pw::Application markUndoLevel {Set VC}\n' ...
    '$_TMP(PW_1) setPhysicalType volumeToCell\n' ...
    'pw::Application markUndoLevel {Change VC Type}\n' ...
    'unset _TMP(PW_1)\n' ...
    '  set _TMP(mode_1) [pw::Application begin CaeExport]\n' ...
    '  $_TMP(mode_1) addAllEntities\n' ...
    '  $_TMP(mode_1) initialize -strict -type CAE ',path,'\n' ...
    '  $_TMP(mode_1) verify\n' ...
    '  $_TMP(mode_1) write\n' ...
    '$_TMP(mode_1) end\n' ...
    'unset _TMP(mode_1)\n']);
```

B.3.7. POINTWISE FARFIELD

```
% This function aims to generate an empty farfield (3D mesh) from a list of
% domains for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID          -> Pointwise ID desired farfield number
%       list        -> List of domains IDs enclosing the geometry of interest
%                      You may also specify 'all' by typing 0
%       optionsFF   -> 'Box' : Box shape farfield
%                      'Cylinder' : Cylindrical shape farfield
%                      'Sphere' : Spherical shape farfield
%       optionsCell -> 'Unstructured' : Unstructured elements in the FF
%                      'Voxel' : Voxel elements in the FF
%       dimensions  -> They depend on the 'optionsFF' specified
%                      if 'Box'      -> 3x2 Matrix : [x_positive x_negative;
%                                                     y_positive y_negative;
%                                                     z_positive z_negative]
%                      if 'Cylinder' -> 3x2 Matrix : [L_positive L_negative;
%                                                     Radius_1   Radius_2;
%                                                     Axis_alignment    0]
%                      if 'Sphere'   -> 1x1 Variable : Radius
%       arrangement -> 1x3 Vector specifying the largest and the shortest
%                      directions e.g. [1 2 3] where 1 is the largest and 3
%                      the lowest (only useful for 'Box' case)
% Outputs:
%       farfieldCommand -> String containing the command to write the
%                          farfield
function farfieldCommand = ...
    pwFarfield(ID,list,optionsFF,optionsCell,dimensions,arrangement)
size = length(list);
string = '[list ';
if size ~= 1
    for i = 1:size
        if i < size
            auxstr = append('$_DM(',num2str(list(i)),') ');
        else
```

```
            auxstr = append('$_DM(',num2str(list(i)),')]]');
        end
        string = append(string,auxstr);
    end
    allBoundary = '';
else
    allBoundary = 'set _TMP(mode_1) [pw::Grid getAll -type pw::Domain]\n';
    string = append(string,'$_TMP(mode_1)]]');
end
switch optionsFF
    case 'Box'
        x_forw = dimensions(arrangement(1),1); x_back = dimensions(arrangement(1),2);
        y_forw = dimensions(arrangement(2),1); y_back = dimensions(arrangement(2),2);
        z_forw = dimensions(arrangement(3),1); z_back = dimensions(arrangement(3),2);
        farfieltype = append(['  $_TMP(mode_2) setFarfieldShapeType Box\n'...
            '  $_TMP(mode_2) setFarfieldLength {',num2str(x_forw),' ...
                ',num2str(x_back),'}\n'...
            '  $_TMP(mode_2) setFarfieldWidth {',num2str(y_forw),' ...
                ',num2str(y_back),'}\n'...
            '  $_TMP(mode_2) setFarfieldHeight {',num2str(z_forw),' ...
                ',num2str(z_back),'}\n']);
    case 'Cylinder'
        L_forw = dimensions(1,1); L_back = dimensions(1,2);
        r_forw = dimensions(2,1); r_back = dimensions(2,2);
        axis = dimensions(3,1);
        switch axis
            case 1
                alignment = 'X';
            case 2
                alignment = 'Y';
            case 3
                alignment = 'Z';
            otherwise
                alignment = 'Automatic';
        end
        farfieltype = append(['  $_TMP(mode_2) setFarfieldShapeType Cylinder\n'...
            '  $_TMP(mode_2) setFarfieldLength {',num2str(L_forw),' ...
                ',num2str(L_back),'}\n'...
            '  $_TMP(mode_2) setFarfieldCapRadii {',num2str(r_forw),' ...
                ',num2str(r_back),'}\n'...
            '  $_TMP(mode_2) setShapeAlignment ',alignment,'\n']);
    case 'Sphere'
        R = dimensions;
        farfieltype = append(['  $_TMP(mode_2) setFarfieldShapeType Sphere\n'...
            '  $_TMP(mode_2) setFarfieldRadius ',num2str(R),'\n']);
    otherwise
        x_forw = dimensions(1,1); x_back = dimensions(1,2);
        y_forw = dimensions(2,1); y_back = dimensions(2,2);
        z_forw = dimensions(3,1); z_back = dimensions(3,2);
        farfieltype = append(['  $_TMP(mode_2) setFarfieldShapeType Box\n'...
            '  $_TMP(mode_2) setFarfieldHeight {',num2str(x_forw),' ...
                ',snum2str(x_back),'}\n'...
            '  $_TMP(mode_2) setFarfieldLength {',num2str(y_forw),' ...
                ',snum2str(y_back),'}\n'...
            '  $_TMP(mode_2) setFarfieldWidth {',num2str(z_forw),' ...
                ',snum2str(z_back),'}\n']);
end
switch optionsCell
    case 'Unstructured'
        meshtype = append('  $_TMP(mode_2) setMeshType Unstructured\n');
    case 'Voxel'
        meshtype = append('  $_TMP(mode_2) setMeshType Voxel\n');
    otherwise
        meshtype = append('  $_TMP(mode_2) setMeshType Unstructured\n');
end

farfieldCommand = append([allBoundary, ...
    'set _TMP(mode_2) [pw::Application begin VolumeMesher ',string,'\n'...
    farfieltype,...
    meshtype,...
    '  $_TMP(mode_2) createGridEntities\n'...
```

```
99      '  $_TMP(mode_2) end\n'...
100     'unset _TMP(mode_1)\n'...
101     'unset _TMP(mode_2)\n'...
102     'set _BL(',num2str(ID),') [pw::Grid getAll -type pw::Block]\n'...
103     'pw::Application markUndoLevel {Automatic Volume Mesh}\n']);
104 end
```


B.3.8. POINTWISE MESH

```
1  % This function aims to generate an unstructured 3D Mesh in pointwise having
2  % already defined an empty volume mesh
3  %       Author  : Gabriel Buendia
4  %       Version : 1
5  % Inputs:
6  %      ID          -> ID of the already existing bulk data entity
7  %      BLoptions   -> Structure containing options for the boundary layer
8  %                     Fields:
9  %                     -maxLayers       -> This default is the maximum number of
10 %                     T-Rex layers of an unstructured block when it is created :
11 %                     Number (Default 0)
12 %                     -fullLayers       -> This default is the minimum number of
13 %                     fully structured T-Rex layers of an unstructured block when
14 %                     it is created :
15 %                     Number (Default 0)
16 %                     -growthRate       -> This default is the growth rate of
17 %                     T-Rex layers of an unstructured block when it is created :
18 %                     Number (Default 1.2)
19 %                     -push             -> This default is the flag for pushing
20 %                     T-Rex attributes onto the connectors and domains of an
21 %                     unstructured block when a it is created :
22 %                     'true' or 'false' (Default false)
23 %                     -solverAttribute  -> Sets the named unstructured solver
24 %                     attribute :
25 %                     'TetPyramidPrismHex'||'AllAndReducePyramids'||
26 %                     'AllAndConvertWallDoms'||'LegacyTetPyramidPrismHex'||
27 %                     'TetPyramid'
28 %                     -firstLayerHeight -> Specifies the BL thickness of the
29 %                     first cell :
30 %                     Number (User specified)
31 %      ISOoptions -> -minLength -> Minimum length of the isotropic tets :
32 %                     Number||'Boundary' (Default Boundary)
33 %                    -maxLength -> Maximum length of the isotropic tets :
34 %                     Number||'Boundary' (Default Boundary)
35 % Outputs:
36 %      volumeMeshCommand -> String containing the command to write the 3D
37 %                           mesh
38 function volumeMeshCommand = pwMesh(ID, BLoptions, ISOoptions)
39 volumeMeshCommand = append(['set _TMP(mode_1) [pw::Application begin ...
       UnstructuredSolver [list $_BL(',num2str(ID),')]]\n' ...
40        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute TRexMaximumLayers ', ...
             num2str(BLoptions.maxLayers), '\n' ...
41        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute TRexFullLayers ', ...
             num2str(BLoptions.fullLayers), '\n' ...
42        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute TRexGrowthRate ',  ...
             num2str(BLoptions.growthRate), '\n' ...
43        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute TRexPushAttributes ', ...
             num2str(BLoptions.push), '\n' ...
44        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute TRexCellType ', ...
             num2str(BLoptions.solverAttribute), '\n' ...
45        '  set _TMP(PW_1) [pw::TRexCondition getByName {Boundary Layer}]\n' ...
46        '  $_TMP(PW_1) setValue ', num2str(BLoptions.firstLayerHeight), '\n' ...
47        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute EdgeMinimumLength ', ...
             num2str(ISOoptions.minLength), '\n' ...
48        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute EdgeMaximumLength ', ...
             num2str(ISOoptions.maxLength), '\n' ...
49        '  $_BL(',num2str(ID),') setUnstructuredSolverAttribute EdgeMaximumGrowthRate ...
             ', num2str(ISOoptions.maxGrowth), '\n' ...
50        '  $_TMP(mode_1) setStopWhenFullLayersNotMet true\n' ...
```

```
        '  $_TMP(mode_1) setAllowIncomplete true\n' ...
        '  $_TMP(mode_1) run Initialize\n' ...
        '$_TMP(mode_1) end\n' ...
        'unset _TMP(mode_1)\n']);
end
```

B

B.3.9. POINTWISE MIRROR

```
% This function aims to generate a mirror copy of all database, connectors
% and domains selected with respect to certain cartesian axis for
% Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       listDB -> Defines the list of database entities (IDs) to be copied
%       listCN -> Defines the list of conectors (IDs) to be copied
%       listDM -> Defines the list of domains (IDs) to be copied
%       axis   -> if 'x' : The reflection plane is yz
%                 if 'y' : The reflection plane is xz
%                 if 'z' : The reflection plane is xy
% Outputs:
%       mirrorCommand -> String containing the command to write the mirror copy
function mirrorCommand = pwMirror(listDB,listCN,listDM,axis)
switch axis
    case 'x'
        axis = [1 0 0];
    case 'y'
        axis = [0 1 0];
    case 'z'
        axis = [0 0 1];
end
sizeDB = length(listDB);
sizeCN = length(listCN);
sizeDM = length(listDM);
string = 'pw::Application setClipboard [list ';
for i = 1:sizeDB
    if ~isnan(listDB(i))
        string = append(string,'$_DB(',num2str(listDB(i)),') ');
    end
end
for i = 1:sizeCN
    if ~isnan(listCN(i))
        string = append(string,'$_CN(',num2str(listCN(i)),') ');
    end
end
for i = 1:sizeDM
    if ~isnan(listDM(i))
        if i < sizeDM
            string = append(string,'$_DM(',num2str(listDM(i)),') ');
        else
            string = append(string,'$_DM(',num2str(listDM(i)),')]\n');
        end
    end
end
mirrorCommand = append(['pw::Application clearClipboard\n', ...
    string, ...
    'pw::Application markUndoLevel Copy\n', ...
    'set _TMP(mode_1) [pw::Application begin Paste]\n', ...
    '  set _TMP(PW_1) [$_TMP(mode_1) getEntities]\n', ...
    '  set _TMP(mode_2) [pw::Application begin Modify $_TMP(PW_1)]\n', ...
    '    pw::Entity transform [pwu::Transform mirroring {',num2str(axis(1)),' ...
     ',num2str(axis(2)),' ',num2str(axis(3)),'} 0] [$_TMP(mode_2) getEntities]\n', ...
    '  $_TMP(mode_2) end\n', ...
    '  unset _TMP(mode_2)\n', ...
    '$_TMP(mode_1) end\n', ...
    'unset _TMP(mode_1)\n']);
end
```

B.3.10. POINTWISE POINTS

```
% This function aims to generate single point for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID -> Pointwise ID desired database number
%       x  -> x coordinate of the point
%       y  -> z coordinate of the point
%       z  -> z coordinate of the point
% Outputs:
%       pointCommand -> String containing the command to write the point
%       in Pointwise
function pointCommand = pwPoints(ID,x,y,z)
pointCommand = append('set _DB(', num2str(ID),[') [pw::Point create]\n' ...
    '$_DB(',num2str(ID),') setPoint {'],num2str(x,'%.6f'),' ', ...
    num2str(y,'%.6f'),' ',num2str(z,'%.6f'),'}\n');
end
```

B.3.11. POINTWISE ROTATION

```
% This function aims to generate a rotation of all entities selected with
% respect to certain fixed point and cartesian axis for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       fixedPoint -> Vector (1x3) defining the fixed point for the rotation
%                     [xRot yRot zRot]
%       rotAxis    -> if 'x' : The rotation axis is x
%                     if 'y' : The rotation axis is y
%                     if 'z' : The rotation axis is z
%       rotAngle   -> Angle to be rotated [deg]
%       option     -> String defining the entities to be rotated
%                     -'allDataBaseandGrid': Rotates all database and grid entities
%                     (connectors and domains) specified in listDB, listCN and list DM
%                     -'allDataBase': Rotates all database entities created
%                     so far withouth list specification
%                     -'allGrid': Rotates all grid entities created so far withouth
%                     list specification
%                     -'allSource': Rotates all source entities created so far withouth
%                     list specification
%                     -'otherDataBase': Rotates all database entities
%                     spcified in the list listO
%                     -'otherConnector': Rotates all connector entities
%                     spcified in the list listO
%                     -'otherDomain': Rotates all domain entities
%                     spcified in the list listO
%                     -'otherSource': Rotates all source entities
%                     spcified in the list listO
%       listO      -> Defines the list of entities (IDs) specified in option 'other...'
%                     to be rotated
%       listDB     -> Defines the list of database entities (IDs) to be rotated
%       listCN     -> Defines the list of conectors (IDs) to be rotated
%       listDM     -> Defines the list of domains (IDs) to be rotated

% Outputs:
%       rotationCommand -> String containing the command to write the rotation
function rotationCommand = ...
    pwRotation(fixedPoint,rotAxis,rotAngle,option,listO,listDB,listCN,listDM)
switch rotAxis
    case 'x'
        rotAxis = [1 0 0];
    case 'y'
        rotAxis = [0 1 0];
    case 'z'
        rotAxis = [0 0 1];
end
```

B

```
switch option
    case 'allDataBaseandGrid'
    sizeDB = length(listDB);
    sizeCN = length(listCN);
    sizeDM = length(listDM);
    string = '  set _TMP(mode_1) [pw::Application begin Modify [list ';
    for i = 1:sizeDB
        if ~isnan(listDB(i))
            string = append(string,'$_DB(',num2str(listDB(i)),') ');
        end
    end
    for i = 1:sizeCN
        if ~isnan(listCN(i))
            string = append(string,'$_CN(',num2str(listCN(i)),') ');
        end
    end
    for i = 1:sizeDM
        if ~isnan(listDM(i))
            if i < sizeDM
                string = append(string,'$_DM(',num2str(listDM(i)),') ');
            else
                string = append(string,'$_DM(',num2str(listDM(i)),')]]\n');
            end
        end
    end
    string2 = '';
    case 'allDataBase'
        string = append(['set _TMP(PW_1) [pw::Database getAll]\n' ...
                  '  set _TMP(mode_1) [pw::Application begin Modify $_TMP(PW_1)]\n']);
        string2 = 'unset _TMP(PW_1)\n';
    case 'allGrid'
        string = append(['set _TMP(PW_1) [pw::Grid getAll]\n' ...
                  '  set _TMP(mode_1) [pw::Application begin Modify $_TMP(PW_1)]\n']);
        string2 = 'unset _TMP(PW_1)\n';
    case 'allSource'
        string = append(['set _TMP(PW_1) [pw::Source getAll]\n' ...
                  '  set _TMP(mode_1) [pw::Application begin Modify $_TMP(PW_1)]\n']);
        string2 = 'unset _TMP(PW_1)\n';
    case 'otherDataBase'
        sizeO = length(listO);
        string = '  set _TMP(mode_1) [pw::Application begin Modify [list ';
        for i = 1:sizeO
            if ~isnan(listO(i))
                if i < sizeO
                    string = append(string,'$_DB(',num2str(listO(i)),') ');
                else
                    string = append(string,'$_DB(',num2str(listO(i)),')]]\n');
                end
            end
        end
        string2 = '';
    case 'otherConnector'
        sizeO = length(listO);
        string = '  set _TMP(mode_1) [pw::Application begin Modify [list ';
        for i = 1:sizeO
            if ~isnan(listO(i))
                if i < sizeO
                    string = append(string,'$_CN(',num2str(listO(i)),') ');
                else
                    string = append(string,'$_CN(',num2str(listO(i)),')]]\n');
                end
            end
        end
        string2 = '';
    case 'otherDomain'
        sizeO = length(listO);
        string = '  set _TMP(mode_1) [pw::Application begin Modify [list ';
        for i = 1:sizeO
            if ~isnan(listO(i))
                if i < sizeO
                    string = append(string,'$_DM(',num2str(listO(i)),') ');
```

```
                else
                    string = append(string,'$_DM(',num2str(listO(i)),')]]\n');
                end
            end
        end
        string2 = '';
    case 'otherSource'
        sizeO = length(listO);
        string = '  set _TMP(mode_1) [pw::Application begin Modify [list ';
        for i = 1:sizeO
            if ~isnan(listO(i))
                if i < sizeO
                    string = append(string,'$_SR(',num2str(listO(i)),') ');
                else
                    string = append(string,'$_SR(',num2str(listO(i)),')]]\n');
                end
            end
        end
        string2 = '';
end
rotationCommand = append([string, ...
    '    pw::Entity transform [pwu::Transform rotation -anchor ...
        {',num2str(fixedPoint(1)),' ',num2str(fixedPoint(2)),' ...
        ',num2str(fixedPoint(3)),'} {',num2str(rotAxis(1)),' ',num2str(rotAxis(2)),' ...
        ',num2str(rotAxis(3)),'} ',num2str(rotAngle),'] [$_TMP(mode_1) ...
        getEntities]\n' ...
    '  $_TMP(mode_1) end\n' ...
    '  unset _TMP(mode_1)\n' ...
    string2]);
end
```

B.3.12. POINTWISE SAVE AND EXIT

```
% This function aims to save the mesh in a Pointwise file and close the
% Pointwise application
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       path -> String that specifies the path for the file to be saved
%       name -> String that specifies the name of the file to be saved
% Outputs:
%       saveandfinishCommand -> String containing the command to save and
%                               close Pointwise
function saveandfinishCommand = pwSaveAndFinish(path,name)
string = append('{',path,'/',name,'.pw}\n');
saveandfinishCommand = append(['pw::Application save ',string ...
                        'pw::Application exit\n']);
end
```

B.3.13. POINTWISE SOURCE

```
% This function aims to generate an empty source (3D refinement region) from
% in any location and shape specified by the user for Pointwise language
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID           -> Pointwise ID desired source number
%       optionsSrc   -> 'Box' : Box shape source
%                       'Cylinder' : Cylindrical shape source
%                       'Sphere' : Spherical shape source
%       startface    -> 'x' : Source start and end faces are defined
%                       following the unitary vector i (x axis)
%                       'y' : Source start and end faces are defined
%                       following the unitary vector j (y axis)
%                       'z' : Source start and end faces are defined
%                       following the unitary vector k (z axis)
```

B

```
%       center       -> Specify the COG of the source in cartesian coords
%                       1x3 Vector : [x_center y_center z_center]
%       dimensions   -> They depend on the 'optionsSrc' specified
%                       if 'Box'      -> 1x3 Vector : [length height width]
%                       if 'Cylinder' -> 1x3 Vector : [radius top_Radius length]
%                                        Note that radius and top_Radius are
%                                        the same for cylinder and different
%                                        if defining a cone
%                       if 'Sphere'   -> 1x3 Vector : [radius base_Angle top_Angle]
%                                        Note that base_Angle -> [0,90] deg and
%                                        top_Angle -> [90,180] deg are used
%                                        to define cut sphere sources
%       distribution -> 'Constant': The begin values will be used
%                        throughout the source
%                       'Parametric' : The begin values will be used at the
%                        minimum parametric limits and the end values will be
%                        used at the maximum parametric limits of the source
%                       'AxisToPerimeter : The begin values will be used along
%                       the axis and the end values will be used at the perimeter
%                       of the source
%                       'CenterToPerimeter : The begin values will be used at
%                       the center and the end values will be used at the perimeter
%                       of the source
%       spec         -> 2x2 Matrix: [spacing_initial spacing_final;
%                                    decay_initial   decay_final]
%                       Note that for the 'constant' case only the initial
%                       values are read
% Outputs:
%       sourceCommand -> String containing the command to write the
%                        source
function sourceCommand = ...
    pwSource(ID,optionsSrc,startface,center,dimensions,distribution,spec)
switch optionsSrc
    case 'Box'
        switch startface
            case 'x'
                R = rotationMatrix(0,270,0);
                T = translationMatrix(-center(3),center(2),center(1));
                len = dimensions(1,1); height = dimensions(1,2); width = dimensions(1,3);
            case 'y'
                R = rotationMatrix(90,0,0);
                T = translationMatrix(center(1),-center(3),center(2));
                len = dimensions(1,2); height = dimensions(1,3); width = dimensions(1,1);
            case 'z'
                R = rotationMatrix(0,0,0);
                T = translationMatrix(center(1),center(2),center(3));
                len = dimensions(1,3); height = dimensions(1,2); width = dimensions(1,1);
        end
        typeDim = append('  $_SR(',num2str(ID),') box  -length ',num2str(len),' ...
            -height ',num2str(height),' -width ',num2str(width));
    case 'Cylinder'
        radius = dimensions(1,1); topRadius = dimensions(1,2); len = dimensions(1,3);
        switch startface
            case 'x'
                R = rotationMatrix(0,270,0);
                T = translationMatrix(-center(3),center(2),center(1));

            case 'y'
                R = rotationMatrix(90,0,0);
                T = translationMatrix(center(1),-center(3),center(2));
            case 'z'
                R = rotationMatrix(0,0,0);
                T = translationMatrix(center(1),center(2),center(3));
        end
        typeDim = append('  $_SR(',num2str(ID),') cylinder -radius ...
            ',num2str(radius),' -topRadius ',num2str(topRadius),' -length ...
            ',num2str(len));
    case 'Sphere'
        radius = dimensions(1,1); baseAngle = dimensions(1,2); topAngle = ...
            dimensions(1,3);
        switch startface
```

```
            case 'x'
                R = rotationMatrix(0,270,0);
                T = translationMatrix(-center(3),center(2),center(1));
            case 'y'
                R = rotationMatrix(90,0,0);
                T = translationMatrix(center(1),-center(3),center(2));
            case 'z'
                R = rotationMatrix(0,0,0);
                T = translationMatrix(center(1),center(2),center(3));
        end
        typeDim = append('  $_SR(',num2str(ID),') sphere -radius ',num2str(radius),' ...
            -baseAngle ',num2str(baseAngle),' -topAngle ',num2str(topAngle));
end
Transform = R*T;
Transform = reshape(Transform,[1 16]);
size = length(Transform);
string = '[list ';
for i = 1:size
    if i < size
        auxstr = append(num2str(Transform(i)),' ');
    else
        auxstr = append(num2str(Transform(i)),']');
    end
    string = append(string,auxstr);
end
sourceCommand = append(['set _TMP(mode_1) [pw::Application begin Create]\n'...
    '  set _SR(',num2str(ID),') [pw::SourceShape create]\n'...
    '  $_SR(',num2str(ID),') setPivot Center\n'...
    typeDim,'\n'...
    '  $_SR(',num2str(ID),') setTransform ',string,'\n' ...
    '$_TMP(mode_1) end\n' ...
    'unset _TMP(mode_1)\n']);
switch distribution
    case 'Constant'
        spacingB = spec(1,1); spacingE = spacingB;
        decayB = spec(2,1); decayE = 0.5;
        typeDist = append('  $_SR(',num2str(ID),') setSpecificationType Constant');
    case 'Parametric'
        spacingB = spec(1,1); spacingE = spec(1,2);
        decayB = spec(2,1); decayE = spec(2,2);
        typeDist = append('  $_SR(',num2str(ID),') setSpecificationType Parametric');
    case 'AxisToPerimeter'
        spacingB = spec(1,1); spacingE = spec(1,2);
        decayB = spec(2,1); decayE = spec(2,2);
        typeDist = append('  $_SR(',num2str(ID),') setSpecificationType ...
            AxisToPerimeter');
    case 'CenterToPerimeter'
        spacingB = spec(1,1); spacingE = spec(1,2);
        decayB = spec(2,1); decayE = spec(2,2);
        typeDist = append('  $_SR(',num2str(ID),') setSpecificationType ...
            CenterToPerimeter');
end
sourceCommand = append([sourceCommand,...
    'set _TMP(mode_1) [pw::Application begin Modify [list $_SR(',num2str(ID),')]]\n'...
    typeDist,'\n'...
    '  $_SR(',num2str(ID),') setBeginSpacing ',num2str(spacingB),'\n'...
    '  $_SR(',num2str(ID),') setEndSpacing ',num2str(spacingE),'\n'...
    '  $_SR(',num2str(ID),') setBeginDecay ',num2str(decayB),'\n'...
    '  $_SR(',num2str(ID),') setEndDecay ',num2str(decayE),'\n'...
    '$_TMP(mode_1) end\n'...
    'unset _TMP(mode_1)\n']);
end
```

B.3.14. POINTWISE SURFACE

```
% This function aims to generate single patched surface from a list of
% curves for Pointwise language. Note that curves do not need to intersect,
% and the intersect tolerance will be specified by tolThres.
```

```
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       ID       -> Pointwise ID desired database number
%       list     -> List of curves IDs that enclose the surface
%       tolBound -> This attribute is the tolerance to use when splitting
%                   and joining the curves specified as the boundaries, for
%                   detecting end to end connections of the boundaries, and
%                   for creating the surface from the boundaries
%       tolThres -> This attribute specifies the percentage of length of
%                   the boundary curves of a created surface that must be
%                   database constrained to automatically set the fitting
%                   entities
% Outputs:
%       curveCommand -> String containing the command to write the surface
function surfaceCommand = pwSurface(ID,list,tolBound,tolThres)
size = length(list);
string = append('  set _DB(',num2str(ID),') [$fitter createPatch ');
for i = 1:size
    if i < size
        auxstr = append('[list $_DB(',num2str(list(i)),')] ');
    else
        auxstr = append('[list $_DB(',num2str(list(i)),')]]\n');
    end
    string = append(string,auxstr);
end
surfaceCommand = append(['set fitter [pw::Application begin SurfaceFit]\n' ...
    '  $fitter setBoundaryTolerance ',num2str(tolBound),'\n' ...
    '  $fitter setFitEntitiesThreshold ',num2str(tolThres),'\n' ...
    string, ...
    '  $fitter run 0\n' ...
    '$fitter end\n']);
end
```

B.3.15. ROTATION MATRIX

```
% This function aims to generate a rotation matrix for a rigid body in
% cartesian coordinates
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       rx: Rotation angle in x axis [deg]
%       ry: Rotation angle in y axis [deg]
%       rz: Rotation angle in z axis [deg]
% Outputs:
%       R : Rotation matrix -> 3x3 Matrix
function R = rotationMatrix(rx,ry,rz)
Rx = [1 0 0 0;
      0 cosd(rx) sind(rx) 0;
      0 -sind(rx) cosd(rx) 0;
      0 0 0 1];
Ry = [cosd(ry) 0 -sind(ry) 0;
      0 1 0 0;
      sind(ry) 0 cosd(ry) 0;
      0 0 0 1];
Rz = [cosd(rz) -sind(rz) 0 0;
      sind(rz) cosd(rz) 0 0;
      0 0 1 0;
      0 0 0 1];
R = Rx*Ry*Rz;
end
```

B.3.16. SCALING MATRIX

```
% This function aims to generate a scaling matrix for a rigid body in
% cartesian coordinates.
```

```
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       sx: Scaling factor in x axis
%       sy: Scaling factor in y axis
%       sz: Scaling factor in z axis
% Outputs:
%       S : Scaling matrix -> 3x3 Matrix
function S = scalingMatrix(sx,sy,sz)
S = [sx 0  0  0;
     0  sy 0  0;
     0  0  sz 0;
     0  0  0  1];
end
```

B.3.17. TRANSLATION MATRIX

```
% This function aims to generate a translation matrix for a rigid body in
% cartesian coordinates
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       tx: Translation vector component in x axis
%       ty: Translation vector component in y axis
%       tz: Translation vector component in z axis
% Outputs:
%       T : Translation matrix -> 3x3 Matrix
function T = translationMatrix(tx,ty,tz)
T = [1 0 0 tx;
     0 1 0 ty;
     0 0 1 tz;
     0 0 0 1];
end
```

B.4. OTHER USEFUL FUNCTIONS

B.4.1. CLUSTER JOB

```
% This function aims to write the .pbs file for launching simulations in the
% TU Delft cluster
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       job -> Structure containing job parameters
%              Fields:
%              -FolderPath: Path to save the file locally
%              -Name: Name asigned to the job in the cluster
%              -Queue: awep-test, awep-fast, awep-small, awep-medium, awep-large
%              -Hours: Time reserved for the job in the cluster
%              -Nodes: Number of nodes to be used
%              -Cores: Number of cores to be used
%              -NodeType: f, k, g, h, i
%              -Solver: Type of solver within OpenFOAM to be used
%              -FinalTime: End of the simulation
%              -Path: Path to specify the working directory in the cluster
function ofClusterJob(job)
fileName = fopen(append(job.FolderPath,'job.pbs'),'w');
if job.Hours > 9
    wallTime = append('#PBS -l walltime=',num2str(job.Hours),':00:00\n');
else
    wallTime = append('#PBS -l walltime=0',num2str(job.Hours),':00:00\n');
end
string = append(['#!/bin/bash\n' ...
                 '#PBS -j oe\n' ...
                 '#PBS -o log.mylog.${PBS_JOBID}\n' ...
```

B

```
                    '#PBS -N ',job.Name,'\n' ...
                    '#PBS -q ',job.Queue,'\n' ...
                    wallTime ...
                    '#PBS -l ...
                        nodes=',num2str(job.Nodes),':ppn=',num2str(job.Cores),':type', ...
                        job.NodeType,'\n' ...
                    '#PBS -m abe\n' ...
                    '#PBS -M G.E.BuendiaVela@student.tudelft.nl\n\n' ...
                    '###################\n' ...
                    '#User Settings\n' ...
                    '###################\n\n' ...
                    'solver=',job.Solver,'\n' ...
                    'runNumber=1\n' ...
                    'finalTime = ',num2str(job.FinalTime),'\n\n' ...
                    'echo Job started on `uname -n` at `date`\n\n' ...
                    'echo "Loading OpenFOAM-v2006"\n' ...
                    'module load mpi/openmpi-1.8.8-gnu\n' ...
                    'module load openfoam/v2006\n\n' ...
                    '# Change to your working directory\n' ...
                    'cd ',job.Path,'\n' ...
                    'echo "Working directory: " ',job.Path,'\n\n' ...
                    '##################\n' ...
                    '# Run OpenFOAM in Parallel\n' ...
                    '##################\n' ...
                    'echo "Starting OpenFOAM job at: " $(date)\n\n' ...
                    '# Decompose for parallel computing\n\n' ...
                    'echo "Renumbering the mesh before decomposition"\n' ...
                    'renumberMesh > log.renumberMesh.1 2>&1\n\n' ...
                    'echo "Checking the mesh"\n' ...
                    'checkMesh -latestTime > log.checkMesh.1 2>&1\n\n' ...
                    'echo "Decomposing the domains"\n' ...
                    'decomposePar -cellDist -force -latestTime > log.decomposePar ...
                        2>&1\n\n' ...
                    '# Create solver log with run number of maximum 10\n' ...
                    'touch log.$runNumber.$solver\n\n' ...
                    '# Run the solver (parallel)\n' ...
                    'which $solver\n' ...
                    'echo "Running " $solver\n' ...
                    '# Actual line runs the openfoam\n' ...
                    'mpirun --hostfile $PBS_NODEFILE $solver -parallel > ...
                        log.$runNumber.$solver 2>&1\n' ...
                    '# Post processing\n' ...
                    'reconstructPar > log.reconstructPar.1 2>&1\n' ...
                    'echo "solution reconstructed"\n\n' ...
                    'touch foam.foam\n\n' ...
                    'rm -rf processor*\n' ...
                    '$PBS_NODEFILE $solver -postProcess -func yPlus -time $finalTime > ...
                        log.$runNumber.$solver.yPlus 2>&1\n' ...
                    'echo "Removing decomposed folders"\n\n' ...
                    'echo "Ending OpenFOAM job at: " $(date)\n']);
fprintf(fileName,string);
fclose(fileName);
end
```

B.4.2. MONITOR SIMULATION SIMPLEFOAM

```
% This function aims to read the data coming directly from OpenFOAM to
% check the convergence of the variables
%       Author  : Gabriel Buendia
%       Version : 1
% Inputs:
%       name -> String specifying the name (and path) of the file to be
%       read
%       numIt -> Number of iterations for the Cauchy residual
% Outputs:
%       forces_c_val -> Matrix containing the forces coeffients at
%                       each timestep (timestepx3)
%       res          -> Vector containing the Cauchy residuals of the
```

```
%                       aerodynamic forces (1x3)
function [forces_c_val,res] = ofScanLogSimpleFOAM(name,numIt)
fileID = fopen(name,'r');
string_full = fscanf(fileID,'%c');
fclose(fileID);
Ux_s = 'Ux, Initial residual = ';
Ux_index = strfind(string_full,Ux_s);
Uy_s = 'Uy, Initial residual = ';
Uy_index = strfind(string_full,Uy_s);
Uz_s = 'Uz, Initial residual = ';
Uz_index = strfind(string_full,Uz_s);
p_s = 'p, Initial residual = ';
p_index = strfind(string_full,p_s);
omega_s = 'omega, Initial residual = ';
omega_index = strfind(string_full,omega_s);
k_s = 'k, Initial residual =';
k_index = strfind(string_full,k_s);
forces_s = 'Sum of forces';
forces_index = strfind(string_full,forces_s)+21;
Ux_val = zeros(1,length(Ux_index));
Uy_val = zeros(1,length(Uy_index));
Uz_val = zeros(1,length(Uz_index));
p_val = zeros(2,length(p_index)/2);
omega_val = zeros(1,length(omega_index));
k_val = zeros(1,length(k_index));
forces_c_val = zeros(3,length(forces_index));
for i = 1:length(Ux_index)
    aux_ind = 1;
    while ~strcmp(string_full(Ux_index(i)+length(Ux_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    Ux_val(i) = ...
        str2num(string_full(Ux_index(i)+length(Ux_s):Ux_index(i)+length(Ux_s)+aux_ind));
end
for i = 1:length(Uy_index)
    aux_ind = 1;
    while ~strcmp(string_full(Uy_index(i)+length(Uy_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    Uy_val(i) = ...
        str2num(string_full(Uy_index(i)+length(Uy_s):Uy_index(i)+length(Uy_s)+aux_ind));
end
for i = 1:length(Uz_index)
    aux_ind = 1;
    while ~strcmp(string_full(Uz_index(i)+length(Uz_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    Uz_val(i) = ...
        str2num(string_full(Uz_index(i)+length(Uz_s):Uz_index(i)+length(Ux_s)+aux_ind));
end
index = 1;
for i = 1:length(p_index)
    aux_ind = 1;
    while ~strcmp(string_full(p_index(i)+length(p_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    if mod(i,2) == 1
        p_val(1,index) = ...
            str2num(string_full(p_index(i)+length(p_s):p_index(i)+length(Ux_s)+aux_ind));
    else
        p_val(2,index) = ...
            str2num(string_full(p_index(i)+length(p_s):p_index(i)+length(Ux_s)+aux_ind));
        index = index+1;
    end
end
for i = 1:length(omega_index)
    aux_ind = 1;
```

```
    while ~strcmp(string_full(omega_index(i)+length(omega_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    omega_val(i) = str2num(string_full(omega_index(i)+length(omega_s):omega_index(i)+ ...
        length(omega_s)+aux_ind));
end
for i = 1:length(k_index)
    aux_ind = 1;
    while ~strcmp(string_full(k_index(i)+length(k_s)+aux_ind),char(44))
        aux_ind = aux_ind+1;
    end
    aux_ind = aux_ind-1;
    k_val(i) = ...
        str2num(string_full(k_index(i)+length(k_s):k_index(i)+length(k_s)+aux_ind));
end

for i = 1:length(forces_index)
    aux_ind = 1;
    for j = 1:3
        switch j
            case 1
                while ~strcmp(string_full(forces_index(i)+length(forces_s)+aux_ind), ...
                    char(32))
                    aux_ind = aux_ind+1;
                end
                aux_ind2 = aux_ind-1;
                aux_ind = aux_ind+1;
            case 2
                while ~strcmp(string_full(forces_index(i)+length(forces_s)+aux_ind), ...
                    char(32))
                    aux_ind = aux_ind+1;
                end
                aux_ind3 = aux_ind-1;
                aux_ind = aux_ind + 1;
            otherwise
                while ~strcmp(string_full(forces_index(i)+length(forces_s)+aux_ind), ...
                    char(41))
                    aux_ind = aux_ind+1;
                end
                aux_ind4 = aux_ind-1;
        end
    end
    forces_c_val(1,i) = ...
        str2num(string_full(forces_index(i)+length(forces_s):forces_index(i)+ ...
        length(forces_s)+aux_ind2));
    forces_c_val(2,i) = ...
        str2num(string_full(forces_index(i)+length(forces_s)+aux_ind2+ ...
        1:forces_index(i)+length(forces_s)+aux_ind3));
    forces_c_val(3,i) = ...
        str2num(string_full(forces_index(i)+length(forces_s)+aux_ind3+ ...
        1:forces_index(i)+length(forces_s)+aux_ind4));
end

figure()
semilogy(Ux_val,'Linewidth',2)
hold on
semilogy(Uy_val,'Linewidth',2)
semilogy(Uz_val,'Linewidth',2)
semilogy(p_val(2,:),'Linewidth',2)
semilogy(omega_val,'Linewidth',2)
semilogy(k_val,'Linewidth',2)
set(gca,'FontSize',14)
xlabel('Iterations','FontSize',14)
ylabel('Residuals','FontSize',14)
leg = legend('U_x','U_y','U_z','p','\omega','k','Location','Best');
leg.NumColumns = 2;
grid on

load("Flight_Conditions.mat")
figure()
```

```
139 semilogy(abs(forces_c_val(3,:))/Q,'Linewidth',2)
140 hold on
141 semilogy(abs(forces_c_val(2,:))/Q,'Linewidth',2)
142 semilogy(abs(forces_c_val(1,:))/Q,'Linewidth',2)
143 set(gca,'FontSize',14)
144 xlabel('Iterations','FontSize',14)
145 ylabel('|Aerodynamic coefficients|','FontSize',14)
146 legend('C_L','C_S','C_D','Location','Best')
147 grid on
148
149 forces_c_val = forces_c_val';
150 fprintf('Drag coefficient\n')
151 resDrag = cauchyResidual(forces_c_val(:,1)/Q,numIt);
152 fprintf('Sideforce coefficient\n')
153 resSide = cauchyResidual(forces_c_val(:,2)/Q,numIt);
154 fprintf('Lift coefficient\n')
155 resLift = cauchyResidual(forces_c_val(:,3)/Q,numIt);
156 res = [resDrag resSide resLift];
157 end
158
159 function res = cauchyResidual(forces,numIt)
160 counter = 0;
161 summatory = 0;
162 while counter ≤ numIt - 1
163     diff = abs(forces(length(forces)-counter)-forces(length(forces)-counter-1));
164     summatory = summatory + diff;
165     counter = counter + 1;
166 end
167 res = summatory/(numIt-1);
168 fprintf('The Cauchy residual is %d\n',res)
169 end
```