Delft University of Technology

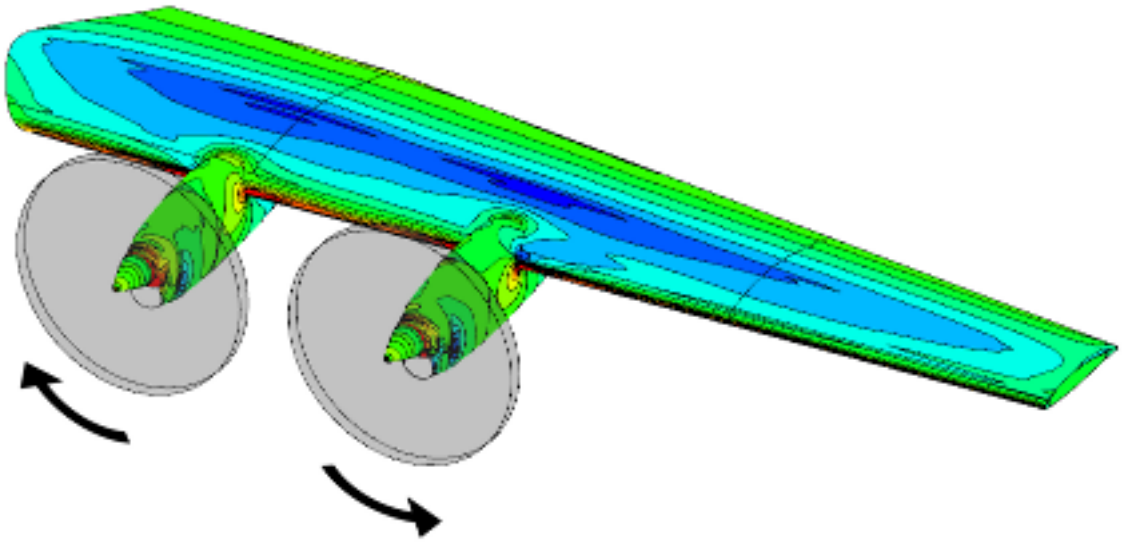
Master of Science Thesis

Belgium

Royal Military Academy

Numerical investigation of propeller-wing interaction effects for a large military transport aircraft

The influence of rotation sense of the propellers



Marilyne Lino

August 27, 2010

Faculty of Aerospace Engineering · Delft University of Technology

Faculty of Aerospace Engineering

Aerodynamics

Numerical investigation of propeller-wing interaction effects for a large military transport aircraft

The influence of rotation sense of the propellers

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Marilyne Lino

August 27, 2010

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Copyright © Aerospace Engineering, Delft University of Technology
All rights reserved.

Delft University Of Technology
Department Of
Aerodynamics

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Numerical investigation of propeller-wing interaction effects for a large military transport aircraft"** by **Marilyne Lino** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: August 27, 2010

Graduation Committee:

Dr. ir. L.L.M. Veldhuis (Supervisor Delft)

Capt. ir. B. Marinus (Supervisor Brussels)

Prof. dr. ir. W. Bosschaerts

Prof. dr. ir. drs. H. Bijl

Summary

The goal of this thesis is to investigate numerically the influence of rotation sense of a propeller on propeller-wing interaction effects for a large multi-engined military transport aircraft using the commercial software FLUENT. One must be able to perform the simulations in a preliminary design stage, even when detailed propeller data is unavailable. From a previously performed literature study it was concluded that the actuator disk theory would be the best model to perform this study.
Although FLUENT provides the FAN boundary condition (mimics an actuator disk) to model propeller induced flow, this option was discarded due to compatibility problems with the earlier FLUENT 6.3 version. Therefore a so called "actuator volume" was created which contains only one column of cells in the axial direction. Thus instead of imposing a jump in pressure, tangential and radial velocity at the trailing edge of the propeller blade; a jump in axial, tangential and radial velocity is imposed at the cell centers of the "actuator volume". The axial, tangential and radial velocities were imposed by loading a User Defined Function (UDF) in FLUENT. This is basically a user-written code in C-programming language that allows the user to create its own numerical models which can be loaded with the FLUENT solver. The equations which were used to calculate these velocities were obtained by slightly modifying the equations for an actuator disk with variable radial loading distribution as obtained by J.T. Conway. The assumptions that were made during the derivation is that one assumes steady, incompressible and inviscid flow. Outside this actuator volume the flow was calculated by using the Reynolds-Averaged Navier-Stokes equations (RANS). The $k - \varepsilon$ model with non-equilibrium wall functions was used to model turbulent flow.
To assess the performance of the UDF and the propeller model in general some validation simulations were performed. To this end a single propeller-nacelle geometry was created based on the NASA SR-1 propeller. Flow parameters such as Mach number, static pressure, etc. were extracted along three different lines of constant radius and compared with the results of a full blade modelling simulation using RANS as obtained by B. Marinus. To investigate the influence of the nacelle on the propeller another UDF was set up, which reads in the inflow velocity at the leading edge of the propeller blade. The equations of the axial, tangential and radial velocity are similar to the original UDF (uniform actuator disk), which assumes a uniform inflow velocity equal to freestream, however the freestream

velocity terms are replaced by the inflow velocity at the blade. Also a UDF was set up which makes use of Phillips' equations in order to investigate the effect of excluding the radial velocity.
From the validation simulations it appeared that the UDF which assumes a uniform inflow velocity equal to freestream and the UDF which makes use of a non uniform inflow velocity (non-uniform actuator disk) produce similar results. Although no radial velocity was imposed by the UDF which makes use of Phillips equations the prediction of the radial velocity was reasonably accurate, suggesting the dominant influence of the nacelle, however the tangential velocity prediction was worse than the other models. Since the non-uniform actuator disk UDF is more complex and hence computationally more expensive, the uniform actuator disk model was chosen as the numerical model to study propeller-wing interaction effects.
To investigate propeller-wing interference a simplified model of a generic large multi-engined military transport aircraft was created. Since stability and control characteristics are of no interest and the focus lies on the influence of the propeller induced flow on the wing, the airplane geometry only consists of a simplified fuselage, a clean wing and engines (nacelles+propellers). Furthermore the geometry is assumed to be symmetric, thus only half of the geometry needs to be modelled. In order to investigate the influence of rotation sense on the aerodynamic performance of the wing three different rotation senses were performed: both engines rotate inboard up, both engines rotate outboard up and the down-between-engines (DBE) rotation.
From the simulations it could be concluded that the inboard up case has the best aerodynamic performance in terms of lift-to-drag ratio and endurance. This is caused by a higher lift production while still maintaining a reasonable overall drag. The outboard up case has the worst aerodynamic performance while the performance of the DBE case is in between the inboard and outboard up case. However it is strongly recommended to investigate the control and stability characteristics of the airplane to see whether the better aerodynamic performance of the inboard up case outweighs the necessary requirements to ensure longitudinal stability of the airplane. If for example larger tail surfaces are necessary to counteract the moments produced in case of one-engine-out, the airplane geometry will be heavier. This can undo the aerodynamic performance benefits.
By inspecting the pressure coefficient plots which were extracted from several span locations along the wing and the spanwise distribution of lift and drag coefficient it followed that the DBE case follows the same trend as the inboard up case from the wing's root till in between both engines, where it displays a behaviour different from the inboard and outboard up case. From here on the behaviour of the DBE configuration is similar to the trend of the outboard up case. The spanwise lift coefficient distribution is similar to the one presented in Malard et al. [2005].
The wing sections which are situated behind the up going blade of the propeller experience an increase in local lift coefficient w.r.t. the prop off case. While the wing sections which are located behind the slipstream of the down going blade experience a decrease in local lift coefficient w.r.t the prop off case. The same observations were made by Moens & Gardarein [2001], Colin et al. [1996], Barber & Nelson [1996] and Zang et al. [2001]. By investigation of the total pressure contours in a plane 1 m upstream and downstream of the wing it can be concluded that the propeller induces a large increase in total pressure which is consumed by the wing. Furthermore by observing the contours for the tangential velocity 1 m up- and downstream of the wing it can be seen that the wing lowers the

tangential velocity. Hence the wing acts as a stator.
It can be concluded that the propeller model fulfills the thesis requirements. It is able to simulate propeller-wing interaction effects in the preliminary design stage. It should be noted however that this model does not simulate the effect of the wing on the propeller blades. The upwash in front of the wing alters the thrust distribution along the span of the propeller. It is therefore recommended to investigate how the upwash effect of the wing on the propeller blades can be implemented in the UDF. Although the propeller model assumes steady, incompressible and inviscid flow, these assumptions do not compromise the results of the simulations since they are only applied to the actuator volume while the rest of the domain was simulated by making use of RANS. Furthermore the validation simulations showed good agreement with the full blade modelling simulations one radius downstream of the disk where compressibility effects are accounted for. However it is recommended to investigate whether the current propeller model can be further improved by incorporating unsteady effects. But despite these flaws the current propeller model produces reasonably accurate results at a low computational cost (resources and time). It can therefore be used in other projects e.g.: effect of engine placement or stability and control issues in case of engine failure.

Acknowledgments

This thesis has been conducted for obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology. The thesis itself was performed at the Belgian Royal Military Academy in close cooperation with TU Delft. I would therefore want to thank Dr. ir. L.L.M. Veldhuis and Prof. dr. ir. W. Bosschaerts to make this collaboration possible and for their continuous support. Furthermore I would like to thank Capt. Ir. B. Marinus, who is my main supervisor at the academy, for his supervision, continuous support and encouragement. I would also want to express my gratitude to Capt. Ir. B. Janssens for all his help concerning Linux and other IT related problems. I also want to take this opportunity to thank all other staff members and fellow students for creating a cosy atmosphere at the work floor. Last but not least I would like to thank my family and close friends for their continuous support, patience, encouragement and love. Without them I would have never made it this far.

Brussels, Belgium
August 27, 2010

Marilyne Lino

Contents

List of Figures

List of Tables

Nomenclature

Latin Symbols

A	Cross-sectional area	$[m^2]$
a	Edge length	$[m]$
A_f	Area of face f	$[m^2]$
B	Number of blades	$[-]$
b	Net flow rate	$[kg/(ms)]$
c	Chord length	$[m]$
C_D	Three-dimensional drag coefficient	$[-]$
c_f	Friction coefficient	$[-]$
C_L	Three-dimensional lift coefficient	$[-]$
C_P	Power Coefficient	$[-]$
C_p	Pressure coefficient	$[-]$
c_p	Specific fuel consumption	$[lb/(lb \cdot hr)]$
C_Q & C'_Q	Torque Coefficient	$[-]$
C_T, C'_T & C''_T	Thrust Coefficient	$[-]$
C_d	Sectional drag coefficient	$[-]$
C_l	Sectional lift coefficient	$[-]$
D	Three-dimensional drag force	$[N]$
D_p	Propeller diameter	$[m]$
dA	Cross-sectional area of an annular element $dA = 2\pi r dr$	$[m^2]$
dQ	Torque of an annular element	$[N \cdot m]$
dr	Width of an annular element	$[m]$

dT	Thrust of an annular element	$[N]$
E	Total internal energy	$[J]$
e	Internal energy	$[J]$
e_a	Approximate relative error	$[-]$
e_{ext}	Extrapolated relative error	$[-]$
F	Resultant force vector	$[N]$
H	Total enthalpy	$[J]$
h	Enthalpy	$[J]$
h	Height	$[m]$
J	Advance ratio	$[-]$
J_f	Face flux	$[-]$
k	Turbulent kinetic energy	$[J]$
L	Three-dimensional lift force	$[N]$
l	Turbulent length scale	$[m]$
L_c	Cut-off length	$[m]$
M	Mach number	$[-]$
n	Number of rotations per unit time (rps)	$[/s]$
N_f	Number of faces	$[-]$
P	Propeller shaft power	$[W]$
p	Static pressure	$[Pa]$
p_t	Total pressure	$[Pa]$
p_{op}	Operating pressure	$[Pa]$
Q	Torque	$[N \cdot m]$
q	Thermal conductivity	$[W/(Km)]$
r	Radius	$[m]$
Re	Reynolds number	$[-]$
S	Surface area	$[m^2]$
S_ϕ	Source of ϕ	$[-]$
S_{ij}	Mean strain rate	$[J]$
T	Propeller net thrust	$[N]$
T_s	Static temperature	$[K]$
T_t	Total temperature	$[K]$
u	Instantaneous velocity component	$[m/s]$
u'	Fluctuation of the velocity component w.r.t. the mean velocity	$[m/s]$
V	Velocity	$[m/s]$
v	Induced velocity	$[m/s]$
Vol	Volume	$[m^3]$
W	Weight	$[N]$

w	Downwash or upwash	$[N]$
x	x-coordinate	$[-]$
y	y-coordinate	$[-]$
z	z-coordinate	$[-]$

Greek Symbols

α	Angle of attack	$[radians]$
α_p	Under-relaxation factor of pressure	$[-]$
β	Blade angle	$[radians]$
Δ	Difference	$[-]$
δ_{ij}	Kronedecker delta	$[-]$
ϵ	Dissipation rate	$[-]$
η	Efficiency	$[-]$
η_j	Propulsive efficiency	$[-]$
Γ	Circulation	$[m^2/s]$
γ	Specific air ratio	$[-]$
Γ_ϕ	Diffusion coefficient of ϕ	$[-]$
κ	von Karman constant	$[-]$
μ	Dynamic viscosity	$[kg/(ms)]$
ν	Kinematic viscosity	$[m^2/s]$
Ω	Angular velocity	$[radians/s]$
Ω	Volume	$[m^3]$
ω	Induced angular velocity	$[radians/s]$
ϕ	Inflow angle	$[radians]$
ϕ	Scalar quantity	$[-]$
ϕ_0	Advance angle	$[radians]$
ρ	Air density	$[kg/m^3]$
σ	Prandtl number	$[-]$
τ	Time scale	$[s]$
τ_w	Wall shear stress	$[Pa]$
τ_{ij}	Stress tensor	$[Pa]$

Subscripts

0	Location far upstream of the propeller plane
1	Location just upstream of the propeller plane

$_2$	Location just downstream of the propeller plane
$_\infty$	Freestream
$_\theta$	Tangential
$_a$	Axial
$_{ext}$	Extrapolated
$_e$	Effective
$_i$	Induced
$_P$	At a certain point P
$_p$	Propeller
$_{ref}$	Reference
$_{rel}$	Relative
$_r$	Radial
$_s$	Location far downstream of the propeller plane, also called the ultimate slipstream
$_t$	Turbulent

Abbreviations

GCI_{fine}	Fine Grid Convergence Index
LE	Leading Edge
RMA	Royal Military Academy
TE	Trailing Edge

Other Symbols

$\bar{}$	Mean

Chapter 1

Introduction

1.1 Background

Today's world is concerned with global warming and other environmental issues. Hence scientists and engineers are investigating more sustainable energy resources to replace fossil fuels. The aerospace industry can of course not lagg behind and is therefore investigating more sustainable ways of propulsion. This has caused a renewed interest in propeller driven aircraft since they are more fuel efficient than jets. Furthermore the prospect of combining propellers with an electrical engine can indeed help solving the current emissions problem.
The strong point of a propeller-driven aircraft is its ability to accelerate a large amount of air at low speeds, which translates into a shorter take-off length and climbing time. These features makes a propeller-driven airplane attractive for military transport operations.
A propeller-driven aircraft in tractor configuration produces a complicated flow field since there is a mutual interaction between the propeller slipstream and other aircraft parts. Due to the rotation of the propeller, there is an increase in axial velocity and a tangential or swirl velocity is induced. Therefore the aircraft geometry must be designed carefully in order to prevent undesired interaction behaviour.
In the preliminary design stage of an airplane, windtunnel tests are too expensive to investigate the best configuration since the geometry will be continuously adapted until the desired design is found. CFD simulations seem to be the best option in this early design stage to find the "optimum" configuration. That is, if the code is inexpensive in terms of time and calculation resources. This is the subject of this thesis: to numerically simulate propeller-wing interaction effects in the preliminary design stage of an airplane. For this thesis project the commercial software FLUENT will be used and the airplane model under consideration is a generic multi-engined military transport aircraft.
Instead of investigating the placement of the nacelles, the influence of rotation sense of the propellers will be investigated. To this end three cases were considered: both engines on a wing rotate inboard up, outboard up and the Down-Between-Engine concept (DBE). The main disadvantage of a propeller-driven aircraft is of course the interaction

of the propeller slipstream with tail surfaces, which can lead to detrimental stability and control characteristics. The DBE concept or counter-rotating propellers can relieve these stability and control issues. This was also acknowledged by Airbus, who designed the A400M: a military transport aircraft with four turboprop engines which implements the down-between-engine concept.
Since this thesis will only focus on the propeller-wing interaction effects, no stability and control analysis will be made. This thesis will consist of three main parts:

- Part I: Theory
- Part II: Validation
- Part III: Propeller-wing interaction study

In part I the terminology concerning propellers will be introduced as well as the mathematical model which will be used to simulate propeller-wing interaction effects. Also general meshing guidelines and FLUENT settings will be discussed.
Part II will show how this mathematical model is transformed into a User-Defined-Function (UDF) in order to incorporate it in FLUENT. Furthermore validation simulations will be performed in order to investigate the performance of this numerical model. To this end a mesh was created for a propeller-nacelle configuration. The simulations performed on this mesh will then be compared with the results of full blade modelling simulations which were conducted by B. Marinus (Marinus [2007]).
Part III will treat the mesh creation and FLUENT settings for the propeller-wing configuration. Furthermore a detailed discussion of the results for the inboard up, outboard up and DBE case will be given and general conclusions will be drawn. But first a clear formulation of the thesis' goal and set-up will be given in section 1.2.

1.2 Thesis objective and set-up

This section will state the main objective of the thesis and which steps will be taken to complete the thesis goal.

Thesis objective

The main objective of the thesis is formulated as follows:

> Investigate the influence of rotation sense of the propellers on propeller-wing interaction effects for a large multi-engined military transport aircraft with special attention to the down-between-engine concept by using the CFD program FLUENT.

FLUENT provides the "FAN" model in order to simulate propeller induced flow. This is done by imposing a jump in total pressure and a swirl velocity. In order to give the user more options than the basic FLUENT models, a so called User-Defined-Function (UDF) can be loaded in FLUENT. This is basically a user-written "code" in C-programming language that allows the user to create its own numerical models which can be loaded with

the FLUENT solver. A UDF can also be used to prescribe the pressure jump, tangential and radial velocity which needs to be imposed at the FAN boundary. However due to compatibility problems with the earlier FLUENT 6.3 version no UDF could be loaded with the FAN boundary condition. Hence the FAN option was discarded and another method was applied to load the UDF which simulates the actuator disk boundary.
There are four additional requirements attached to the main thesis objective:

1. The code (UDF) must already be applicable in a preliminary design stage.
2. The calculation cost has to be kept at a minimum while aiming for high accuracy of the results.
3. The code (UDF) must be developer-friendly (readable and maintainable for the unfamiliar programmer).
4. Only the aerodynamic performance is of interest. The influence of the interaction effects on stability and control parameters is not required.

The first requirement implies that the code must also be applicable when detailed propeller data e.g. $C_L(r)$, $C_D(r)$, ... is unavailable. This data is unknown during the preliminary design stage of an aircraft, especially when one decides to use a new engine (not yet (fully) developed).

Thesis set-up

In order to perform an accurate numerical simulation, a calculation domain or mesh and suitable mathematical model need to be set up. Once the mathematical model is obtained, it needs to be implemented in FLUENT. This is done by writing a User Defined Function (UDF), which is a user-written program that can be dynamically loaded with the FLUENT solver to customize FLUENT to the specific modelling needs of the user. The mesh is created using the program Gambit and/or T-Grid. In order to assess the accuracy and shortcomings of the developed mathematical model, several validation simulations must be performed. The validation process compares the results of the developed numerical model with the results of a full blade modelling simulation. The full propeller blade modelling simulations make use of the Reynolds-Averaged Navier-Stokes equations (RANS) and were obtained by B. Marinus (Marinus [2007]). If the results of the validation simulations are accurate enough, one is able to apply this numerical model to a simplified aircraft model to investigate the interaction between propeller installation and wing. The aircraft model will consist only of a fuselage, clean wing (no high-lift devices incorporated) and engine installation. The tail surfaces can be neglected since the goal of the thesis only comprises the investigation of the influence of interaction effects on the aerodynamic flow field without looking at stability and control characteristics. The rotation direction of the propellers will be varied in order to study the importance of rotation sense on propeller-wing interaction effects. Special attention will be paid to the down-between-engine concept: two propeller engines, placed on the same wing, which rotate in opposite direction. A schematical representation of the set-up of the thesis can be found in Figure 1.1.

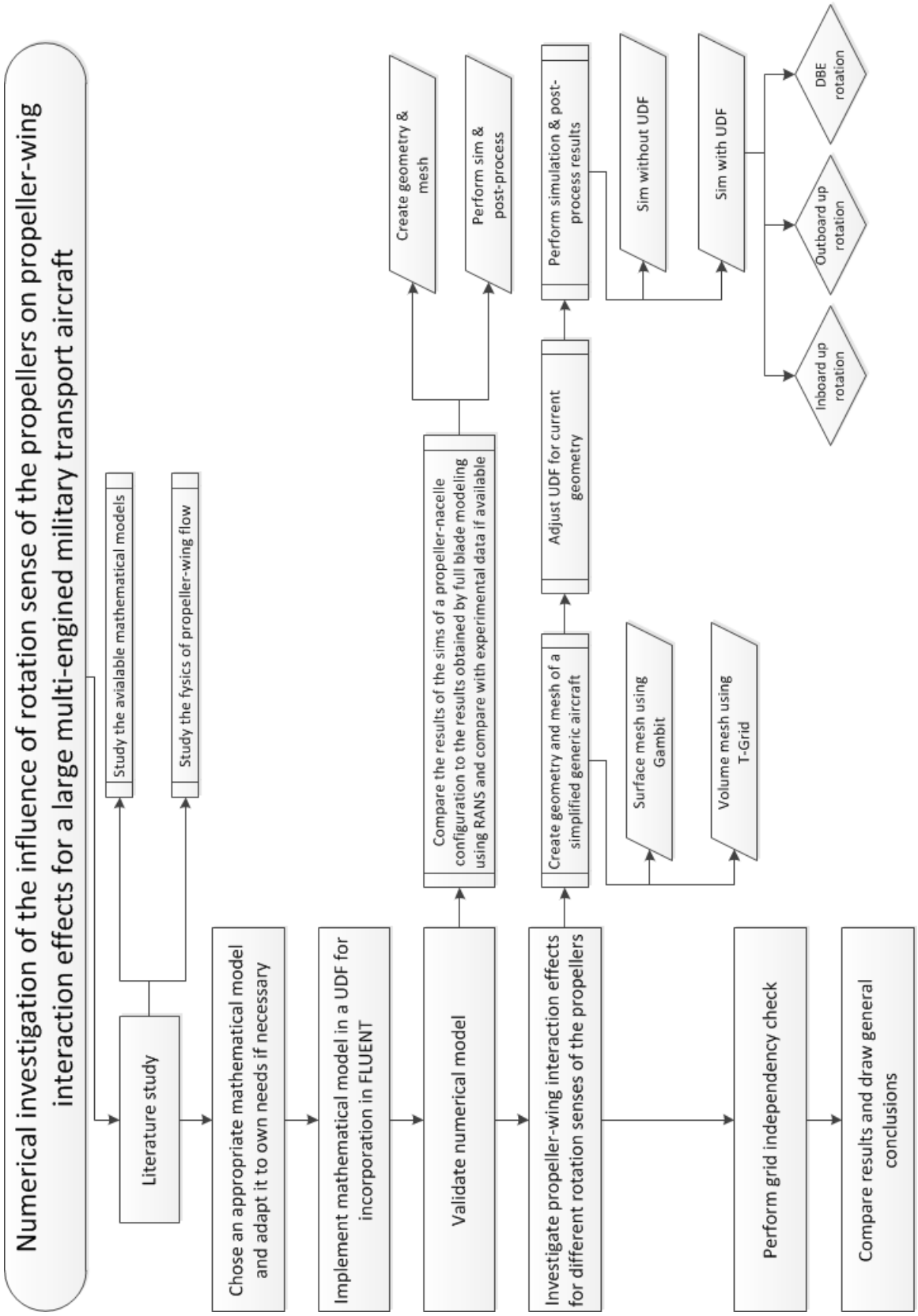


Figure 1.1: Thesis set-up.

Part I: Theory

This part of the thesis elaborates on the background theory, which is necessary to understand the chapters, which treat the validation of the numerical model and the propeller-wing interaction study. As stated in section 1.2 the goal of the thesis is to investigate the influence of rotation sense of the propellers on propeller-wing interaction effects for a large multi-engined military transport aircraft. Since stability and control characteristics are of no interest the airplane geometry can be simplified: it only consists of a simplified fuselage, a clean wing and engines (nacelles+propellers). Since the aircraft geometry is assumed to be symmetric, only half of the geometry needs to be modelled. A representation of the geometry of the airplane can be found in Figure 1.2 and a detailed description of the creation of this geometry and its mesh can be found in section 5.1.

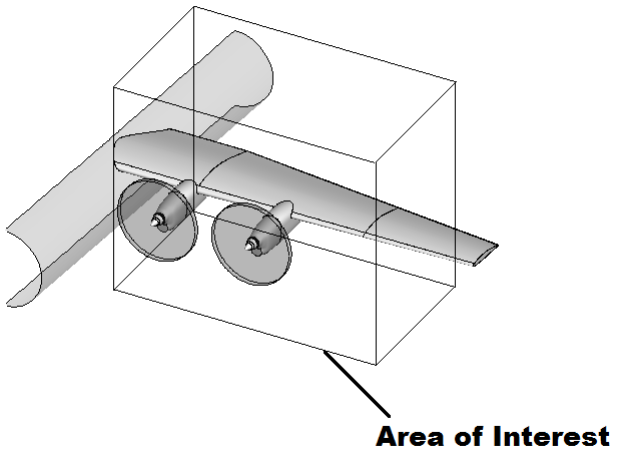


Figure 1.2: Simplified aircraft model and area of interest.

The main interest of this study is to evaluate how the propeller slipstream changes the aerodynamic performance of the wing for different rotation directions of the propellers.

Hence the main area of interest is indicated by the box in Figure 1.2. The main actors influencing propeller-wing interaction effects are the wing, the propellers and nacelles. Section 2.1 will give a clear description of the mutual interaction effects between these components. Then a clear introduction to propeller terminology will be given such that the different mathematical theories that were developed to model propeller induced flow can be introduced in section 2.3.
The last chapter of this part of the thesis is named "Numerical considerations" and consists of 2 sections. The first section will introduce the terminology concerning meshing and general guidelines for grid creation. The last section will discuss the settings available in FLUENT which were used during the simulations.

Chapter 2

Background theory

2.1 Propeller, nacelle and wing interaction effects

The flow field of an isolated propeller or stand-alone propeller and an installed propeller (nacelle and/or wing included) differs significantly. The upwash induced by the wing influences the flow field of the propeller blades, which will be discussed in section 2.1.1. Most propeller-driven airplanes use nacelles to incorporate the engine installation aerodynamically in the design. The presence of the nacelle influences the flow field of the propeller while the propeller itself also influences the flow around the nacelle. Section 2.1.2 will discuss how the interaction effects cause loads on the propeller hub, while the overall effects on and caused by the nacelle are shortly discussed in section 2.1.3. At the same time, the flow field of a wing emersed in the propeller slipstream has different aerodynamic properties than a clean wing as will be discussed in section 2.1.4.

2.1.1 Effect of the wing on the propeller.

The upwash induced by the airplane's wing changes the local angle of attack of the propeller blades similar to an uninstalled propeller subjected to an angle of attack (see Figure 2.1). The effective velocity experienced by the downward rotating blade is increased by the wing's upwash, which leads to a local angle of attack increase. This results in an increase in elemental lift and blade loading, which augments the thrust and torque on the blade.

The upward rotating blade on the other hand experiences a decrease in the local angle of attack leading to a decreased elemental lift and blade loading. Thus the upward rotating blade experiences a decrease in thrust and torque.

2.1.2 Effect of the propeller-wing configuration on the propeller hub.

The effects of the propeller-wing configuration on the hub of a propeller are caused by the interaction of the propeller and the wing's circulation. The propeller hub loads that are

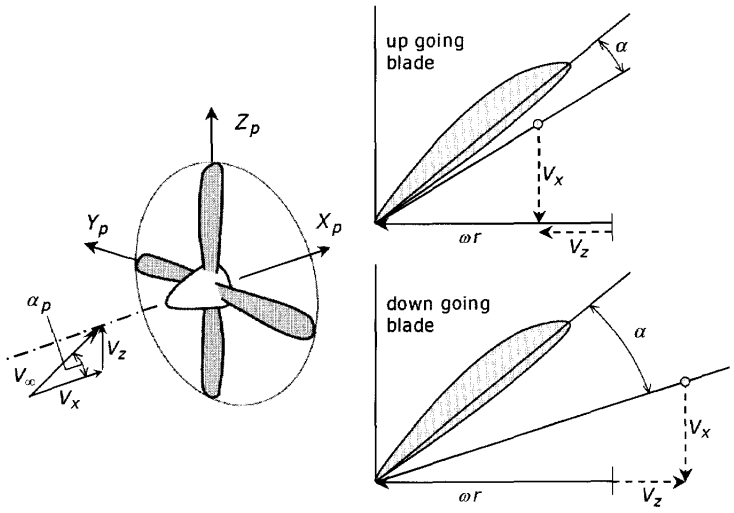


Figure 2.1: Effect of the wing's upwash on the propeller blades. "Veldhuis [2005]"

caused by these interaction effects were investigated by Witkowsky and Lee (Witkowsky et al. [1988]). Their main results are summarized below .

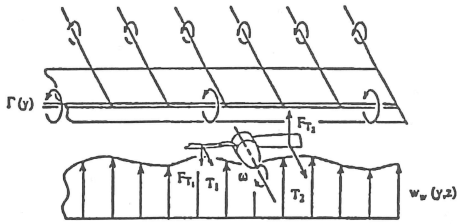


Figure 2.2: Influence of the wing upwash on the propeller blades of a tractor configuration. "Witkowsky et al. [1989]"

The effects of the wing's upwash on the propeller blades was discussed previously in section 2.1.1. The resulting hub loads can be found in Figure 2.2. Since the thrust is increased on the downward going blade and decreased on the upward rotating blade, there will be an imbalance in thrust causing a yawing moment on the hub.

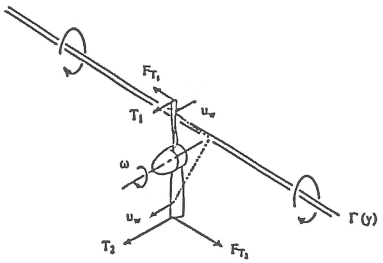


Figure 2.3: Influence of the wing circulation on the propeller blades of a tractor configuration. "Witkowsky et al. [1989]"

The hub side force and pitching moment are caused by the influence of the wing circulation on the axial velocity at the propeller plane. At the blade below the wing there is a

decrease in axial velocity, leading to an increase in angle of attack of the blade and its loads. Concurrently, the blade which is situated above the wing experiences an axial velocity increase leading to a decrease in angle of attack of the blade and its loads. This difference in axial velocity for the blades situated below and above the wing causes an imbalance in thrust force which result in a hub pitching moment. The resulting imbalance in tangential force causes a hub side force, as can be seen in Figure 2.3.

As follows from the previous discussion the hub loads affect the aircraft's stability and structural integrity. This thesis will not investigate these effects so they are not included in the numerical simulations.

2.1.3 Effect of the nacelle on the propeller and vice-versa.

The presence of the nacelle changes the velocity field with respect to the one of an isolated propeller. For today's common used nacelle types, the local velocity distribution is higher than freestream, changing to freestream toward the tip of the propeller blade as can be seen from Figure 2.4. This non-uniform axial velocity distribution is caused by the blockage effect introduced by the presence of the nacelle and will affect the thrust distribution along the blade.

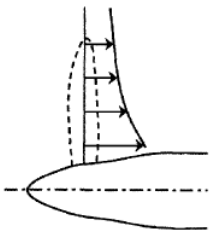


Figure 2.4: Influence of the nacelle on the velocity over the propeller blades. "Veldhuis [2005]"

There is also a strong effect of the propeller on the nacelle leading to propeller-nacelle interaction effects due to an increase in axial and tangential velocity downstream of the propeller. The velocity changes inside the propeller slipstream increases the drag of the nacelle. This drag can even be further increased due to flow separation (Borst [1981]).

This thesis investigates the effects of different propeller rotations on the wing, therefore from all above mentioned effects only the non-uniform axial velocity distribution will be taken into account; since this has a strong influence on the flow downstream of the propeller.

2.1.4 Effect of the propeller slipstream on the wing

By using numerical and experimental methods, one came to the conclusion that the propeller-wing interaction effects for certain tractor propeller configurations resulted in significant wing drag reduction (Witkowsky et al. [1988], Witkowsky et al. [1989] and Veldhuis [2005]). This section will elaborate how this drag reduction can be obtained.

Consider an infinite wing, which does not produce lift at zero angle of attack. The propeller swirl produces regions of propeller upwash and downwash on the wing. The section of the wing which is situated in the upwash region experiences a local increase in angle of attack leading to an increase in local lift coefficient and thus an augmentation of the lift. Furthermore the force vector is tilted forward which leads to a positive lift and negative drag component or thrust (See Figure 2.5).

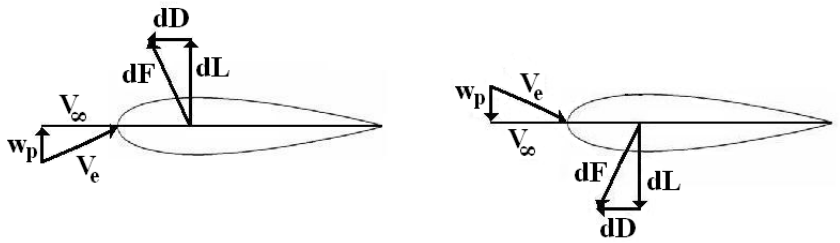


Figure 2.5: Local forces on a wing section located inside the propeller upwash region (left) and the propeller downwash region(right).

If the wing is located in the downwash region of the propeller, the local angle of attack is decreased. This leads to a decrease in local lift coefficient and the force vector is tilted backwards producing a negative lift component and a negative drag component.

Suppose now that the infinite wing is given a positive angle of attack. The generated local force vector in the upwash region is augmented and rotated forward as before leading to a positive lift and reduction in drag. In the propeller downwash region the local force vector is diminished and rotated backward producing positive lift and drag. The net induced drag loading on the wing is still less since the forward rotating force vector is increased and the backward rotating force vector decreased.

If the wing is finite, the wing spanload affects the drag reduction which is obtained by propeller-wing interaction. For finite wings the wing lift loading is larger inboard than outboard. This causes a larger drag reduction for inboard up rotating propellers compared to outboard up rotating propellers. A change in a wing's spanloading will thus either augment or diminish the spanload gradients resulting in better or worse performance characteristics when compared to the original wing. Since spanload gradients are the strongest near the wing tip the positive effect of the propeller slipstream will be stronger if the propeller is placed there (Miranda & Brennan [1986]).

As discussed earlier, there is an increase in axial velocity inside the propeller slipstream and a large amount of swirl. Since the axial velocity inside the slipstream is higher than on the clean wing, there is an increase in dynamic pressure. This increase in dynamic pressure does not change the local lift and drag coefficients when based on the local flow conditions inside the slipstream. However it does increase the local lift and drag coefficients when based on the undisturbed flow conditions. For a propeller at zero angle of attack the axial velocity distribution is symmetrical with respect to the thrust axis. Therefore the change in lift and drag coefficients will also behave symmetrical (See Figure 2.6). The axial velocity does however differ in radial direction, thus the increase in dynamic pressure depends on the vertical position of the propeller with respect to the wing.

As discussed previously, the tangential velocity in the slipstream causes an angle of attack increase or decrease depending on the fact whether the blade is going upwards or

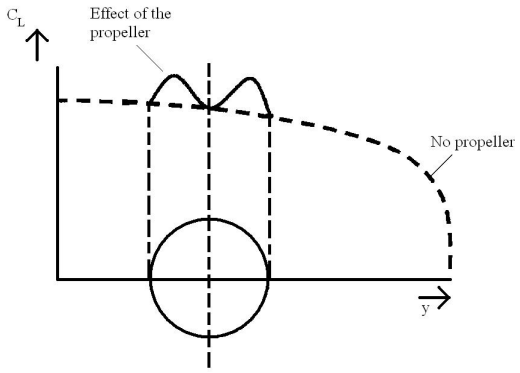


Figure 2.6: Effect of Axial Velocity on the the wing lift coefficient. "Veldhuis [2005]"

downwards in front of the wing, see Figure 2.7 .

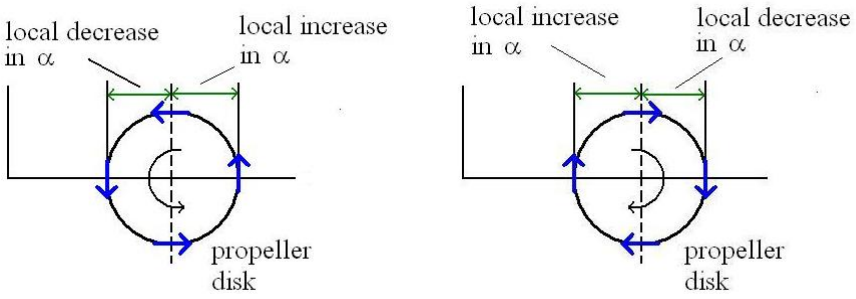


Figure 2.7: Effect of swirl for Outboard up rotation (left) and Inboard up rotation (right).

An increase in local angle of attack leads to an increase in local lift coefficient while a decrease in local angle of attack leads to a decrease in local lift coefficient. The effect of the tangential velocity on the local lift coefficient of the wing is anti-symmetrical and is depicted in Figure 2.8.

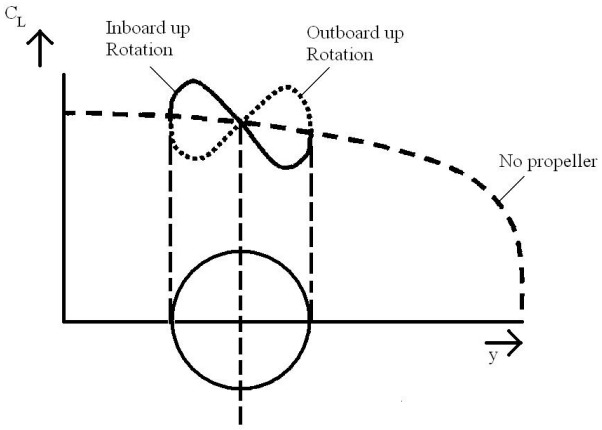


Figure 2.8: Effect of Tangential Velocity on the the wing lift coefficient. "Veldhuis [2005]"

It should be noted that Figure 2.6 and 2.8 only show the effect of the propeller on the wing inside the slipstream. However the propeller also influences the wing regions outside the slipstream. This is caused by the distortion of the vorticity sheet, when leaving the wing. Combining the effects of the axial and tangential velocity and also taking the influences on the wing parts outside the slipstream into account leads to Figure 2.9.

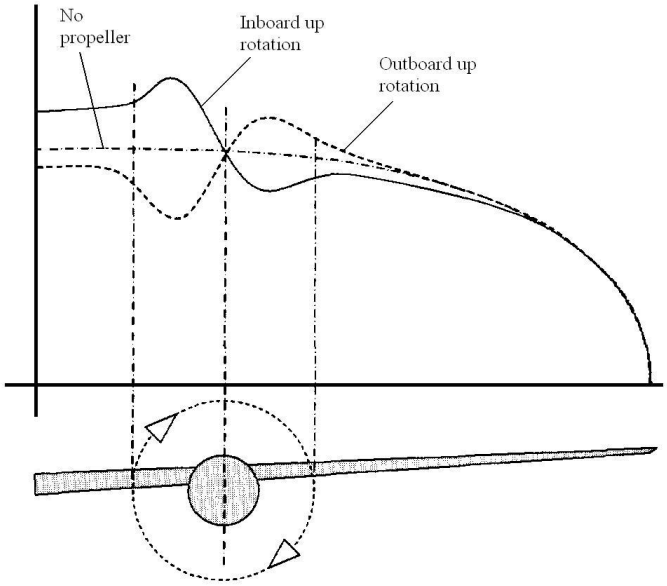


Figure 2.9: Effect of propeller slipstream on the wing lift coefficient. "Veldhuis [2005]"

The modification of the wing's local angle of attack caused by the propeller swirl can be predicted by the actuator disk model as was pointed out by Colin et al. [1996]. Furthermore Feng [2001] showed that the actuator disk approach manages to model propeller-wing interaction effects reasonably accurate, even when the fuselage, nacelle and flaps are included in the geometry.
It should be noted that the influence of the propeller slipstream on the wing's wake will not be investigated.

2.2 General definitions and terminology

This chapter will shortly discuss the basic terminology and definitions used in propeller aerodynamics, which will be used throughout this document.
For several classical propeller theories such as the blade element method, a propeller blade is divided into several blade elements. Consider such a blade element extending over an elementary height dr along the radius at a distance r_p from the axis of rotation as shown in Figure 2.10.

The propeller has an angular velocity Ω_p. The blade element has a chord length, c, while the angle between the chord line of the blade element and the plane of rotation is β. The advance angle, ϕ_0, is defined by the freestream velocity, V_∞, and the rotational velocity as follows:

$$\phi_0 = \arctan \frac{V_\infty}{\Omega_p r_p} = \frac{V_\infty}{2\pi n r_p} \tag{2.1}$$

with: n=number of revolutions per unit time

As can be seen from Eq (2.1) the advance angle decreases from root to tip for specified values of freestream and angular velocity.

If one neglects the induced velocities due to the vortex system, the velocity of the blade element relative to the air is given by Eq (2.2)

$$V_{rel} = \sqrt{V_\infty^2 + (\Omega_p r_p)^2} \tag{2.2}$$

Applying the concept of airfoil theory to the propeller, it is apparent that there must be a circulation of flow around the blades in order to produce lift. This circulation is caused by the presence of a vortex line bound to the blade and running from root to tip. According to the vortex theory, a vortex line cannot begin nor end abruptly. So this bound vortex continues as two free vortices, one emerging from the tip and the other one from the root of the blade. The induced velocity v_i can be broken down in 3 components:

- the axial induced velocity v_a
- the tangential induced velocity v_θ
- the radial induced velocity v_r

Compared to the axial and tangential induced velocity, the radial induced velocity is small and is therefore often neglected in calculations. The radial induced velocity is not displayed in Figure 2.10 since it is in the direction perpendicular to the page. The induced velocity shifts the flow under an angle α_i, which is called the induced angle of attack. So the actual flow acting on the blade element makes an angle ϕ with the plane of rotation. This so called aerodynamic advance angle or inflow angle is given by Eq (2.3).

$$\phi = \arctan \frac{V_\infty + v_a}{\Omega_p r_p - v_\theta} \tag{2.3}$$

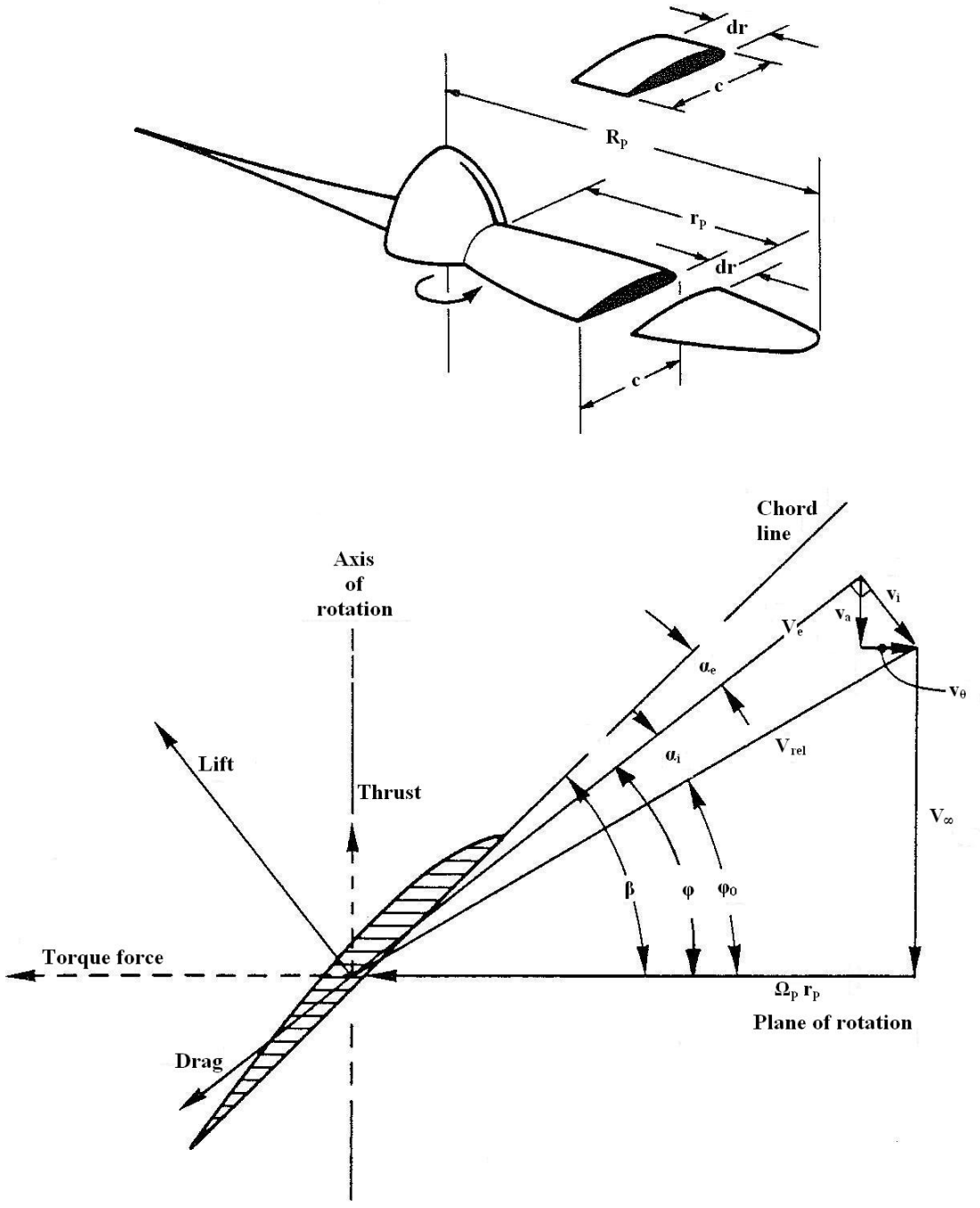


Figure 2.10: Terminology of a blade element. "Jansen [1991]"

The effective velocity can be calculated as:

$$V_e = \sqrt{(V_\infty + v_a)^2 + (\Omega_p r_p - v_\theta)^2} \tag{2.4}$$

The difference between the blade angle and the inflow angle is the effective angle of attack α_e:

$$\alpha_e = \beta - \phi \tag{2.5}$$

The drag works in the direction of the effective velocity and the lift acts perpendicular to it. The thrust and torque force can be found by decomposing the lift and drag forces respectively onto the axis of rotation and the plane of rotation. The thrust T and torque Q can also be expressed in terms of the following non-dimensional coefficients:

- Thrust Coefficients

$$C_T = \frac{T}{\rho n^2 D^4} \tag{2.6}$$

$$C'_T = \frac{T}{\rho V_\infty^2 D^2} \tag{2.7}$$

$$C''_T = \frac{T}{\frac{1}{2}\rho V_\infty^2 S} \tag{2.8}$$

- Torque Coefficients

$$C_Q = \frac{Q}{\rho n^2 D^5} \tag{2.9}$$

$$C'_Q = \frac{Q}{\rho V_\infty^2 D^3} \tag{2.10}$$

- Power Coefficient

$$C_P = \frac{P}{\rho n^3 D^5} \tag{2.11}$$

Equations (2.6) and (2.9) are referenced to the propeller characteristics such as the number of revolutions per unit time and the propeller diameter while Equations (2.7) and (2.10) are referenced to the freestream velocity and propeller diameter.

Eq (2.8) can be compared to the aircraft's coefficients since they both use the wing area and dynamic pressure as reference variables, but it is not commonly used nowadays.

Eq (2.7) and Eq (2.10) can also be expressed in terms of the non-dimensional Advance Ratio J, which is a measure of the advance of the propeller per revolution:

$$J = \frac{V_\infty}{nD} \tag{2.12}$$

Substitution of the expression for J into Eq (2.7) and Eq (2.10) leads to the following expressions:

$$C'_T = \frac{C_T}{J^2} \tag{2.13}$$

$$C'_Q = \frac{Q}{J^3} = \frac{C_Q}{J^2} \tag{2.14}$$

Dimensional analysis shows that these coefficients are a function of advance ratio, blade angle, Mach number and Reynolds number. So $C_T = f(J, \beta, M, Re)$ and $C_Q = f(J, \beta, M, Re)$.

Other important parameters of the propeller are:

- Propeller net efficiency

$$\eta = \frac{TV_\infty}{P} \tag{2.15}$$

  with: P = Propeller shaft power

  Eq (2.15) can also be expressed in terms of the non-dimensional coefficients, C_T, C_Q and J as follows:

$$\eta = \frac{C_T J}{C_P} \tag{2.16}$$

- Propeller solidity, which is the ratio of blade area to disk area:

$$\sigma = \frac{Bc}{2\pi r_p} \tag{2.17}$$

  with:
  B = Number of blades
  c = Chord of a blade element

2.3 The actuator disk theory

Over the past centuries several mathematical models have been developed to predict propeller induced flow. From a previously performed literature study Lino [2010] it was concluded that from all these theories, the actuator disk theory would comply the best with the stated thesis objectives. Therefore this chapter will be dedicated to the derivation of the actuator disk theory. The axial momentum theory was developed in the $19th$ century and is a very simple model since it neglects the swirl velocity inside the slipstream. Therefore the axial momentum theory was later updated into the general momentum theory, which does take the effect of propeller swirl into account. This chapter consists of 3 subsections and is based on references Durand & Glauert [1935], De Lathouder [1948], Ruijgrok [1996] , Carlton [2007], Phillips [2004] and Jansen [1991]. The first subsection will treat the simple axial momentum theory to make the reader familiar with the basic concepts of the actuator disk theory. The second subsection will derive the general momentum theory which does include the tangential and radial induced velocity in its derivation. However no general expression for the radial induced velocity was obtained in this section. The last subsection will introduce a general momentum theory which takes the tangential and radial induced velocity into account and will give general expressions to determine the axial, tangential and radial induced velocities. This theory for variable blade loading was developed by J.T Conway (Conway [1995]) and will be used in the numerical simulation to study propeller-wing interaction effects.

2.3.1 Axial Momentum theory

The axial momentum theory was developed by Rankine in 1865 for marine propellers. This simple theory is based on the following assumptions:

- The flow is considered to be incompressible and inviscid.
- The propeller acts as a disk with an infinite number of blades (actuator disk).
- The flow through the disk is uniform and there is no rotational motion in the slipstream.
- The propeller produces a pressure jump across the disk equal to the thrust per unit area of the disk.

It is assumed that the pressure perturbations induced by the propeller vanish far upstream and far downstream of the propeller. Since the propeller does not influence the flow far upstream of the propeller, its velocity is that of the freestream. A propeller sucks in air through its disk area, so the velocity just in front of the disk must be greater than freestream. According to Bernoulli's Equation the pressure just in front of the propeller must then be less than ambient and the continuity equation states that the area of the streamtube must decrease (See Figure 2.11). Since the disk adds mechanical energy to the flow which passes through it, the velocity far behind the propeller will become larger than freestream. Because of the increase in energy of the flow just behind the propeller, the pressure will be greater than ambient, but its velocity will be the same as just in front of the disk. Far behind the disk, in the slipstream, the pressure will return to ambient pressure.

Let V_{a_s} be the axial velocity in the ultimate slipstream, where the pressure has returned to its original value. If ρ is the density of the flow and A_s the cross-sectional area of the slipstream, then the thrust of the propeller is given by:

$$T = \rho V_{a_s} A_s (V_{a_s} - V_\infty) \tag{2.18}$$

Eq (2.18) states that the thrust of the propeller is equal to the time rate of change of axial momentum. The thrust is also represented by the increase of pressure at the disk:

$$T = A_p(p_2 - p_1) = A_p \Delta p \tag{2.19}$$

Application of Bernoulli's equation upstream and downstream of the disk leads to the following set of equations:

- Upstream of the disk:
$$p_0 + \frac{1}{2}\rho V_\infty^2 = p_1 + \frac{1}{2}\rho V_{a_p}^2 \tag{2.20}$$
- Downstream of the disk:
$$p_2 + \frac{1}{2}\rho V_{a_p}^2 = p_0 + \frac{1}{2}\rho V_{a_s}^2 \tag{2.21}$$

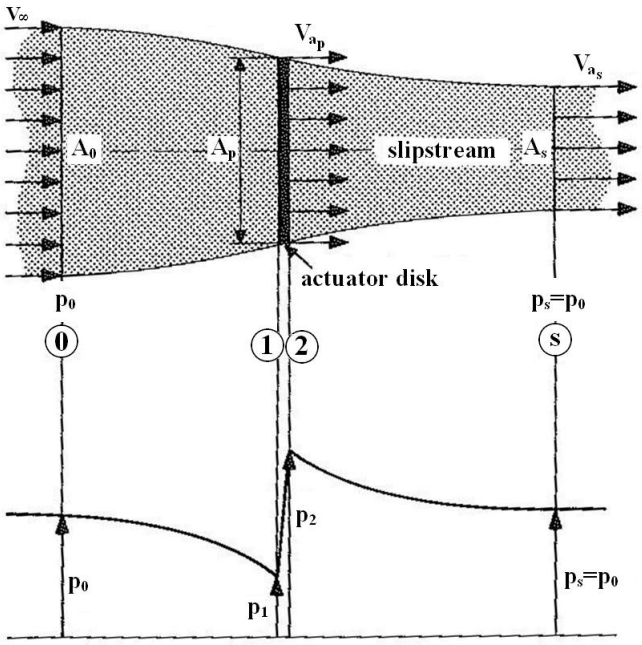


Figure 2.11: Velocity and static pressure distribution according to the axial momentum theory. "Ruijgrok [1996]"

By subtracting Eq (2.21) from Eq (2.20), one finds the following results:

$$\frac{1}{2}\rho(V_\infty^2 - V_{a_s}^2) = p_1 - p_2 = -\Delta p \tag{2.22}$$

$$\Rightarrow \Delta p = \frac{1}{2}\rho(V_{a_s}^2 - V_\infty^2) \tag{2.23}$$

Filling in Eq (2.23) into Eq (2.19) gives the following formula:

$$T = \frac{1}{2}\rho A_p(V_{a_s}^2 - V_\infty^2) \tag{2.24}$$

The continuity condition states that:

$$V_{a_p}A_p = V_{a_s}A_s \tag{2.25}$$

Combining Eq (2.18), Eq (2.24) and Eq (2.25) leads to the following result:

$$T = \rho A_p V_{a_p}(V_{a_s} - V_\infty) = \frac{1}{2}\rho A_p(V_{a_s}^2 - V_\infty^2)$$

$$\Rightarrow V_{a_p}(V_{a_s} - V_\infty) = \frac{1}{2}(V_{a_s}^2 - V_\infty^2)$$

Or

$$V_{a_p} = \frac{1}{2}\frac{(V_{a_s}^2 - V_\infty^2)}{(V_{a_s} - V_\infty)} \tag{2.26}$$

$$V_{a_p} = \frac{1}{2}(V_\infty + V_{a_s}) \tag{2.27}$$

Eq (2.27) states that the axial velocity at the actuator disk is the arithmetic mean of the axial freestream velocity V_∞ and the axial slipstream velocity V_{a_s}. The axial velocity is the sum of the freestream velocity and the induced axial velocity, v_a. The axial velocity at the actuator disk and in the ultimate slipstream are then given by Eq (2.28) and Eq (2.29), respectively.

$$V_{a_p} = V_\infty + v_{a_p} \tag{2.28}$$

$$V_{a_s} = V_\infty + v_{a_s} \tag{2.29}$$

Combining Eq (2.27), Eq (2.28) and Eq (2.29) leads to formula (2.30).

$$v_{a_p} = \frac{1}{2} v_{a_s} \tag{2.30}$$

From Eq (2.30) one can conclude that the induced axial velocity at the propeller plane is half the value of the axial induced velocity in the slipstream.

The propeller power P is equal to the increase of kinetic energy of the air mass flow rate:

$$P = \frac{1}{2}\rho A_p V_{a_p}(V_{a_s}^2 - V_\infty^2) \tag{2.31}$$

From Eq (2.15) the propulsive efficiency can then be expressed as:

$$\eta = \frac{\rho V_{a_s} A_s (V_{a_s} - V_\infty) V_\infty}{\frac{1}{2}\rho A_p V_{a_p}(V_{a_s}^2 - V_\infty^2)}$$

$$\Rightarrow \eta = \frac{2V_\infty}{V_{a_s} + V_\infty} \tag{2.32}$$

Eq (2.32) states that the propulsive efficiency increases as the slipstream velocity decreases.

Filling in Eq (2.27) into Eq (2.32) leads to the following expression:

$$\eta = \frac{V_\infty}{V_{a_p}} \tag{2.33}$$

The efficiency can also be expressed in terms of the thrust. Combining Eq (2.18) and Eq (2.25) leads to the following equation:

$$T = 2\rho V_{a_p} A_p (V_{a_p} - V_\infty) \tag{2.34}$$

Eq (2.34) can be rewritten as an ordinary second order equation in V_{a_p} as follows:

$$2\rho A_p V_{a_p}^2 - 2\rho A_p V_\infty V_{a_p} - T = 0 \tag{2.35}$$

Solving Eq (2.35) for V_{a_p} gives:

$$V_{a_p} = \frac{V_\infty}{2} + \sqrt{\frac{V_\infty^2}{4} + \frac{T}{2\rho A_p}} \tag{2.36}$$

Using Eq (2.36), the induced axial velocity can be determined by means of Eq (2.37).

$$v_{a_p} = \sqrt{\frac{V_\infty^2}{4} + \frac{T}{2\rho A_p}} - \frac{V_\infty}{2} \tag{2.37}$$

Filling in the obtained expression for V_{a_p} into Eq (2.33) leads to the following expression:

$$\eta = \frac{V_\infty}{\frac{V_\infty}{2} + \sqrt{\frac{V_\infty^2}{4} + \frac{T}{2\rho A_p}}} \tag{2.38}$$

Or

$$\eta = \frac{2}{1 + \sqrt{1 + \frac{T}{\frac{1}{2}\rho V_\infty^2 A_p}}} \tag{2.39}$$

Eq (2.38) expresses the propulsive efficiency in function of the propeller area, thrust and freestream velocity. However keep in mind that this propulsive efficiency is the theoretical upper limit of the attainable propulsive efficiency since the axial momentum theory does not include any losses such as the turbulent mixing and the rotational energy of the air lost in the slipstream.

2.3.2 General Momentum theory

The axial momentum theory which was derived in the previous section was based on the assumption that a propeller can be replaced by an actuator disk which produces a sudden increase in pressure at the disk without a change in velocity. The rotational motion in the slipstream was neglected. In order to have a more realistic representation of the flow, the rotational motion has to be included in the analysis. R.E. Froude improved the axial momentum theory by including the rotational velocity to the slipstream induced by the propeller, while the axial and radial components remain unchanged. This so called General Momentum Theory will be elaborated in this section.
Since fluid rotation is taken into account in this model, the assumption of a uniform pressure distribution is no longer valid when a finite amount of thrust is supposed to be generated by the propeller. Therefore the pressure inside the slipstream tube is allowed to depend on the axial and radial position. When the air flows through the propeller, it is subjected to an angular velocity. However far upstream of the propeller plane, the angular velocity is zero, hence the angular velocity must also be zero everywhere inside the streamtube upstream of the propeller plane. Define r_p as the radial distance to an annular element of the propeller disk. The terms V_a and V_r are then respectively the axial

and radial components of the fluid velocity. The pressure just in front of the propeller is called p, while Δp is the increase in pressure just behind the propeller associated with an angular velocity ω_2. Let p_s be the pressure, V_{a_s} the axial velocity and ω_s the angular velocity in the ultimate slipstream at a radial distance r_s from the axis of rotation. Eq (2.40) then states the condition for constancy of angular momentum of the fluid as it passes down the slipstream. A representation of the slipstream tube can be found in Figure 2.12.

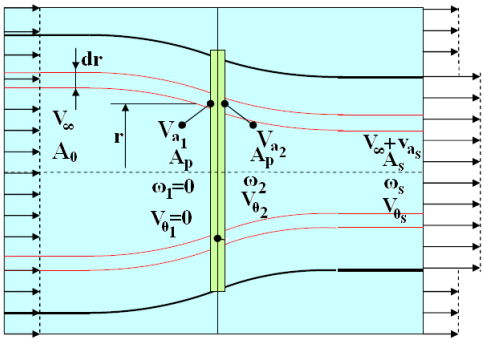


Figure 2.12: Representation of a streamtube. "Sanchez-Caja [2009]"

$$\omega_s r_s^2 = \omega_2 r_p^2 \tag{2.40}$$

Because of Bernoulli's theorem, the axial velocity increases from far upstream of the propeller (region 0) to the near upstream side of the propeller(region 1). Since the static pressure is equal to ambient pressure far upstream of the propeller and decreases to a value below ambient at the near upstream side of the propeller. The pressure at the near downstream side of the propeller plane (region 2) is higher than ambient and decreases back to ambient pressure in the ultimate slipstream (region s). Because of this increase in axial velocity, the slipstream tube contracts radially. The amount of contraction of the streamtube increases with increasing distance downstream of the propeller. From Eq (2.40) it follows that the angular velocity inside the slipstream will increase when moving further downstream of the propeller.

Applying the continuity equation to any segment of the propeller disk leads to Eq (2.41).

$$V_{a_1} = V_{a_2} \tag{2.41}$$

Since $\omega_1 = 0$ the term V_{a_1} can be expressed as follows:

$$V_{a_1} = V_\infty + v_{a_p} \tag{2.42}$$

By combining Eq (2.41) and Eq (2.42) the axial velocity near the downstream side of the propeller can be expressed with formula (2.43).

$$V_{a_2} = V_\infty + v_{a_p} \tag{2.43}$$

Applying Bernoulli's equation upstream and downstream of the propeller leads to:

- Upstream of the propeller

$$p_{t0} = p_0 + \frac{1}{2}\rho V_\infty^2 = p_1 + \frac{1}{2}\rho(V_{a_p}^2 + V_{r_p}^2) \tag{2.44}$$

- Downstream of the propeller

$$p_{t_s} = p_2 + \frac{1}{2}\rho(V_{a_p}^2 + V_{r_p}^2 + \omega_2^2 r_p^2) = p_s + \frac{1}{2}\rho(V_{a_s}^2 + \omega_s^2 r_s^2) \tag{2.45}$$

$$= p_1 + \Delta p + \frac{1}{2}\rho(V_{a_p}^2 + V_{r_p}^2 + \omega_2^2 r_p^2) = p_s + \frac{1}{2}\rho(V_{a_s}^2 + \omega_s^2 r_s^2) \tag{2.45b}$$

Taking the difference of Eq (2.44) and Eq (2.45b) gives:

$$p_s - p_0 + \frac{1}{2}\rho\left(V_{a_s}^2 + \omega_s^2 r_s^2 - V_\infty^2\right) = p_2 - p_1 + \frac{1}{2}\rho\omega_2^2 r_p^2 \tag{2.46}$$

The difference in total pressure can also be expressed by Eq (2.47).

$$p_{t_s} - p_{t0} = \Delta p + \frac{1}{2}\rho\omega_2^2 r_p^2 \tag{2.47}$$

Eq (2.47) states that the increase in total pressure through the propeller is larger than the pressure increase Δp. One must include the kinetic energy of the rotational motion imparted to the fluid by the torque of the propeller. The difference in static pressure can be determined as follows:

$$p_0 - p_s = \frac{1}{2}\rho(V_{a_s}^2 - V_\infty^2) + \frac{1}{2}\rho(\omega_s^2 r_s^2 - \omega_2^2 r_p^2) - \Delta p \tag{2.48}$$

From Eq (2.48) it follows that the pressure in the slipstream p_s is normally less than the ambient pressure p_0.

In the ultimate slipstream, the flow becomes independent of the axial position. Therefore the continuity and momentum equations in radial coordinates for this steady, incompressible, inviscid and axisymmetric flow can be expressed in the ultimate slipstream as follows:

$$\frac{\partial(r_s V_{r_s})}{\partial dr_s} = 0 \tag{2.49}$$

$$\rho\left(V_{r_s}\frac{\partial V_{r_s}}{\partial dr_s} - \frac{V_{\theta_s}^2}{r_s}\right) = -\frac{\partial p_s}{\partial r_s} \tag{2.50}$$

$$\rho\left(V_{r_s}\frac{\partial V_{\theta_s}}{\partial dr_s} + \frac{V_{r_s}V_{\theta_s}}{r_s}\right) = 0 \tag{2.51}$$

$$\rho V_{r_s}\frac{\partial V_{a_s}}{\partial dr_s} = 0 \tag{2.52}$$

Since there is no radial velocity at the outer edge of the ultimate slipstream, Eq (2.49) reduces to:

$$V_{r_s} = 0 \tag{2.53}$$

Formula (2.53) states that there is no slipstream contraction in the ultimate slipstream. Eq (2.51) and Eq (2.52) comply with Eq (2.53) while Eq (2.50) can be rewritten as:

$$\frac{\partial p_s}{\partial r_s} = \rho \frac{V_{\theta_s}^2}{r_s} = \rho \omega_s^2 r_s \tag{2.54}$$

The pressure at the outer edge of the slipstream, where $r_s = R_s$ is equal to the freestream pressure, p_∞. Using this boundary condition and integrating Eq (2.54) gives the following relation:

$$p_s = p_\infty - \frac{\rho \omega_s^2}{2}\left(R_s^2 - r_s^2\right) \quad \text{or} \quad p_s - p_\infty = -\frac{\rho \omega_s^2}{2}\left(R_s^2 - r_s^2\right) \tag{2.55}$$

Substituting formula (2.55) into Eq (2.46) leads to Eq (2.56):

$$\rho \omega_s^2 \left(r_s^2 - \frac{R_s^2}{2}\right) + \frac{1}{2}\rho\left(V_{a_s}^2 - V_\infty^2\right) = p_2 - p_1 + \frac{1}{2}\rho \omega_2^2 r_p^2 \tag{2.56}$$

By applying conservation of mass Eq (2.57) is obtained.

$$\pi r_p^2 \rho \left(V_\infty + v_{a_p}\right) = \pi r_s^2 \rho V_{a_s} \tag{2.57}$$

Rewriting Eq (2.57) in terms of r_s leads to the following expression for r_s:

$$r_s = \sqrt{\frac{V_\infty + v_{a_p}}{V_{a_s}}}\, r_p \tag{2.58}$$

Substituting above expression for r_s into Eq (2.56) and Eq (2.40) leads to Eq (2.59) and Eq (2.60) respectively.

$$\rho \omega_s^2 \left(r_p^2 - \frac{R_p^2}{2}\right) \frac{V_\infty + v_{a_p}}{V_{a_s}} + \frac{1}{2}\rho\left(V_{a_s}^2 - V_\infty^2\right) = p_2 - p_1 + \frac{1}{2}\rho \omega_2^2 r_p^2 \tag{2.59}$$

$$\omega_2 r_p = \frac{V_\infty + v_{a_p}}{V_{a_s}} r_p \omega_s \tag{2.60}$$

Substituting Eq (2.60) into formula (2.59) leads to Eq (2.61).

$$\rho \omega_s^2 \left(r_p^2 - \frac{R_p^2}{2}\right) \frac{V_\infty + v_{a_p}}{V_{a_s}} + \frac{1}{2}\rho\left(V_{a_s}^2 - V_\infty^2\right) = p_2 - p_1 + \frac{1}{2}\rho \left(\frac{V_\infty + v_{a_p}}{V_{a_s}}\right)^2 r_p^2 \omega_s^2 \tag{2.61}$$

Eq (2.61) can be rewritten in terms of the static pressure difference as follows:

$$p_2 - p_1 = \rho \omega_s^2 \left[\left(1 - \frac{V_\infty + v_{a_p}}{V_{a_s}}\right) r_p^2 - \frac{R_p^2}{2}\right] \frac{V_\infty + v_{a_p}}{V_{a_s}} + \frac{1}{2}\rho\left(V_{a_s}^2 - V_\infty^2\right) \tag{2.62}$$

By extending the analysis of the previous section, one obtains the following equation of axial momentum for the propeller:

$$T = \int \rho V_{a_s}(V_{a_s} - V_\infty) dA_s - \int (p_0 - p_s) dA_s \tag{2.63}$$

Filling in Eq (2.55) above formula can be rewritten as:

$$T + \int_{r_s=0}^{R_s} \frac{\rho \omega_s^2}{2} \left(R_s^2 - r_s^2\right) 2\pi r_s dr_s = \int_{r_s=0}^{R_s} \rho V_{a_s}(V_{a_s} - V_\infty) 2\pi r_s dr_s \tag{2.64}$$

Integrating Eq (2.64) leads to to Eq (2.65).

$$T = \rho \pi R_s^2 \left(V_{a_s}(V_{a_s} - V_\infty) - \frac{R_s^2 \omega_s^2}{4} \right) \tag{2.65}$$

By using Eq (2.58) formula (2.66) is obtained.

$$T = \rho \pi R_p^2 \left[(V_\infty + v_{a_p})(V_{a_s} - V_\infty) - \left(\frac{V_\infty + v_{a_p}}{V_{a_s}} \right)^2 \omega_s^2 R_p^2 \right] \tag{2.66}$$

The total thrust can also be expressed by means of Eq (2.67).

$$T = \int_{A_p} (p_2 - p_1) dA_p = \int_{r_p=0}^{R_P} (p_2 - p_1) 2\pi r_p dr_p \tag{2.67}$$

Integrating Eq (2.67) yields:

$$T = \frac{1}{2} \rho \pi R_p^2 \left[V_{a_s}^2 - V_\infty^2 - \left(\frac{V_\infty + v_{a_p}}{V_{a_s}} \right)^2 \omega_s^2 R_p^2 \right] \tag{2.68}$$

Combining Eq (2.66) and Eq (2.68) gives the following result.

$$T = \frac{1}{2} \left(V_{a_s}^2 - V_\infty^2 \right) = (V_\infty + v_{a_p})(V_{a_s} - V_\infty) \tag{2.69}$$

Solving Eq (2.69) for the axial velocity in the slipstream V_{a_s} lead to Eq (2.70).

$$V_{a_s} = V_\infty + 2 v_{a_p} \tag{2.70}$$

Eq (2.70) is identical to the expression for V_{a_s}, obtained by the axial momentum theory (section 2.3.1).

The propeller torque can be determined by means of Eq (2.71).

$$dQ = \int_{A_s} \omega_s r_s^2 d\dot{m} = \int_{r_s=0}^{R_s} \omega_s r_s^2 \rho V_{a_s} 2\pi r_s dr_s \tag{2.71}$$

Integrating Eq (2.71) and using Eq (2.58) leads to Eq (2.72)

$$dQ = \pi\rho\frac{(V_\infty + v_{a_p})^2}{2V_{a_s}}\omega_s R_p^4 \tag{2.72}$$

The brake power of the propeller can be expressed with formula (2.73)

$$dQ\Omega_p = \int_{A_s}\left(h_s - h_\infty + \frac{V_{a_s}^2 + (\omega_s r_s)^2 - V_\infty^2}{2}\right)d\dot{m} \tag{2.73}$$

For uniform, incompressible flow formula (2.73) can be rewritten as:

$$dQ\Omega_p = \int_{r_s=0}^{R_s}\left(\frac{p_s}{\rho} - \frac{p_\infty}{\rho} + \frac{V_{a_s}^2 + (\omega_s r_s)^2 - V_\infty^2}{2}\right)\rho V_{a_s} 2\pi r_s dr_s \tag{2.74}$$

Integrating above equation and applying Eq (2.58) yields the following expression:

$$dQ\Omega_p = \frac{1}{2}\pi R_p^2\rho(V_\infty + v_{a_p})\left(V_{a_s}^2 - V_\infty^2\right) \tag{2.75}$$

By combining Eq (2.72) and Eq (2.75) the following expression is found for the determination of the angular velocity in the slipstream ω_s:

$$\omega_s = \frac{V_{a_s}\left(V_{a_s}^2 - V_\infty^2\right)}{\Omega_p R_p^2(V_\infty + v_{a_p})} \tag{2.76}$$

By substituting Eq (2.70) and Eq (2.76) into Eq (2.69) the total thrust can also be expressed with formula (2.77).

$$T = 2\pi R_p^2\rho(V_\infty + v_{a_p})v_{a_p}\left[1 - \frac{2(V_\infty + v_{a_p})v_{a_p}}{\Omega_p^2 R_p^2}\right] \tag{2.77}$$

The propeller brake power can be expressed as:

$$T = \Omega_p Q = 2\pi R_p^2\rho(V_\infty + v_{a_p})^2 v_{a_p} \tag{2.78}$$

The propeller net efficiency can be determined by means of Eq (2.15), which can be rewritten as Eq (2.79) after substitution of formulae (2.77) and (2.78).

$$\eta = \frac{V_\infty}{V_\infty - \frac{2V_\infty v_{a_p}}{\Omega_p^2 R_p^2}} \tag{2.79}$$

Eq (2.77) is an ordinary second order equation in terms of $(V_\infty + v_{a_p})v_{a_p}$. Solving this equation for $(V_\infty + v_{a_p})v_{a_p}$ leads to formula (2.80).

$$(V_\infty + v_{a_p})v_{a_p} = \frac{\Omega_p^2 R_p^2}{4}\left(1 \pm \sqrt{1 - \frac{4T}{\pi\rho\Omega_p^2 R_p^4}}\right) \tag{2.80}$$

Solving Eq (2.80) for the induced axial velocity term v_{a_p} leads to Eq (2.81).

$$v_{a_p} = \sqrt{\frac{V_\infty^2}{4} + \frac{\Omega_p^2 R_p^2}{4}\left(1 \pm \sqrt{1 - \frac{4T}{\rho\pi\Omega_p^2 R_p^4}}\right)} - \frac{V_\infty}{2} \tag{2.81}$$

Since the induced axial velocity has to be finite and positive if the angular velocity of the propeller becomes large, Eq(2.81) reduces to:

$$v_{a_p} = \sqrt{\frac{V_\infty^2}{4} + \frac{\Omega_p^2 R_p^2}{4}\left(1 - \sqrt{1 - \frac{4T}{A_p\rho\Omega_p^2 R_p^2}}\right)} - \frac{V_\infty}{2} \tag{2.82}$$

By comparing Eq (2.82) with Eq (2.37) it can be seen that the slipstream rotation increases the induced axial velocity. Figure 2.13 displays the difference in propeller net efficiency and induced axial velocity when using the axial and general momentum theory. The displayed error in the plots is given with respect to the general momentum equations. This implies that the induced axial velocity is underestimated while the propeller net efficiency is overestimated when slipstream rotation is neglected.

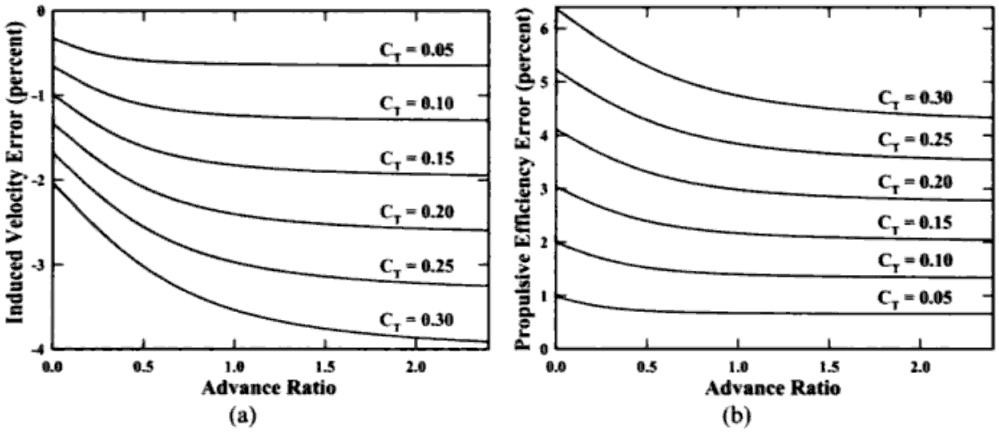


Figure 2.13: Errors introduced by neglecting slipstream rotation for the induced axial velocity (a) and propeller net efficiency (b). "Phillips [2004]"

2.3.3 Analytical model for an actuator disk with variable radial loading distribution

This section will derive the expressions for the axial, tangential and radial velocities at the outflow plane of a general radial loaded actuator disk and is split into two subsections. The first subsection will treat the derivation of the expressions of the axial, tangential and radial induced velocities as obtained by J.T. Conway (Conway [1995]). This derivation is only a short summary of these formulae, for a more detailed discussion the reader is referred to Conway [1995]. The second subsection will show how the analytical solution as obtained by J.T. Conway are used to find expressions for the axial, tangential and radial velocities which will be implemented in the UDF in order to simulate propeller induced flow.

Original expressions for the axial, tangential and radial velocities induced by the actuator disk

The Biot-Savart law linearly relates the instantaneous velocity fields to the instantaneous vorticity distribution for an unsteady, incompressible flow. This implies that the time-averaged velocity fields are given by the steady flow induced by the time-average of the vorticity distribution. The time-averaged velocity field induced by a propeller can be approximated by an actuator disk model if the bound circulation of the propeller blades is represented by straight lifting lines which lie in a plane. The time-average of the bound and shed vorticity of a propeller is the superposition of the vorticity distribution shown in Figure 2.14 . These two self-conservering vortex systems (ring and longitudinal vortex system) consist of four vortex distributions which are responsible for the axial, tangential and radial induced velocities:

1. A vortex tube consisting of ring vortices.
2. A constant-strength hub vortex along the axis of symmetry.
3. A distribution of radial vorticity on the actuator disk.
4. A surface distribution of vorticity on the slipstream surface normal to the ring vortices and equal in strength to the hub vortex.

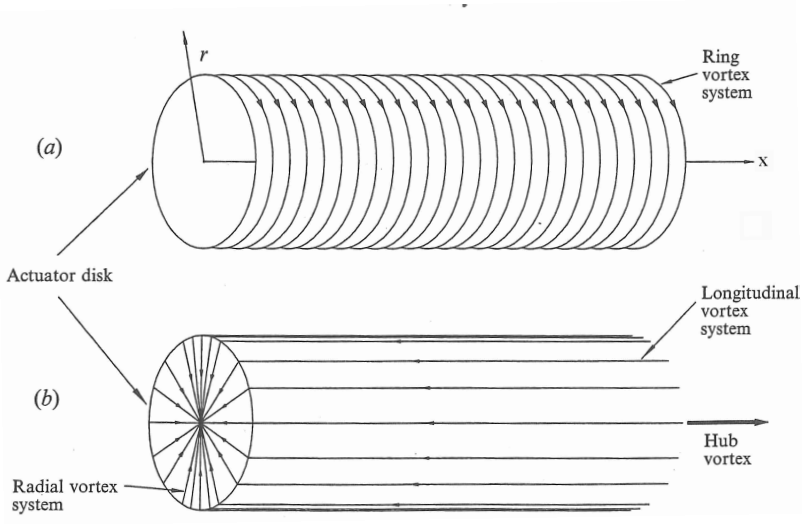


Figure 2.14: Vortex systems for a uniformly loaded actuator disk. (a) Ring vortex system. (b) Longitudinal vortex system. "Conway [1995]"

The ring vortex system consists only of the ring vortices and is directly responsible for the slipstream contraction and all axial and radial induced velocities. The induced axial velocity is derived by constructing the velocity and potential fields induced by a ring vortex as integrals over the allowed values of the separation constant of the eigensolutions of Laplace's equation in cylindrical coordinates. The ring vortex solutions are then combined to give the solution for a general loaded actuator disk as triple integrals of these eigensolutions. The integrals are first integrated in radial direction to give analytical solutions

in terms of elliptic integrals for the velocities and of vector potential by representing the radial variation as an even polynomial. One then obtains Eq (2.83), which is also given in the elementary momentum theory.

$$v_a(r,0) = \frac{v_a(r,\infty)}{2} \tag{2.83}$$

Eq (2.83) and all following equations are written in cylindrical coordinates (r, θ, x) with the x-axis pointing downstream and aligned with the propeller axis. The origin is positioned at the intersection of the x-axis actuator plane. Because of axisymmetric considerations, the θ coordinate is neglected so that everything is expressed in terms of (r, x) coordinates. Eq (2.83) states that the induced axial velocity at the actuator disk is half the value of the induced axial velocity far downstream (in the ultimate wake).

The longitudinal vortex system consists of the hub vortex, the radial vorticity and the vorticity on the slipstream surface. By using the axial symmetry of the distributions together with Stoke's theorem one can show that v_θ is the only non-zero induced velocity component for this system, regardless of axial position and contraction of the slipstream and is given by Eq (2.84) and Eq (2.85). The term R_p in these two equations stands for the tip radius of the actuator disk (i.e. the tip radius of the to be modelled propeller)

- Inside the slipstream ($x > 0$ and $r < R_p$)

$$v_\theta(r,x) = \frac{\Gamma(r)}{2\pi r} \tag{2.84}$$

- Outside the slipstream ($x < 0$ or $r > R_p$)

$$v_\theta(r,x) = 0 \tag{2.85}$$

Thus the induced tangential velocity is zero outside the slipstream and does not vary with axial position within the slipstream. The blade-bound circulation, $\Gamma(r)$, is given by Eq (2.86).

$$\Gamma(r) = \frac{4\pi v_a(r,0)\,(V_\infty + v_a(r,0))}{\Omega_p} \tag{2.86}$$

where the term Ω_p represents the angular velocity of the propeller.

The thrust on an annular element of the actuator disk is calculated from the pressure discontinuity and is given by Eq (2.87).

$$dT(r) = 2\pi\rho r \left[2v_a(r,0)(V_\infty + v_a(r,0)) - \frac{v_\theta^2(r,\infty)}{2}\right] \tag{2.87}$$

The radial induced velocity for a general loaded actuator has a very complex integral form. However the expression for the radial induced velocity for an elliptic axial velocity inflow at the actuator disk is very simple and is given by Eq (2.88).

$$v_r(r,0) = -\frac{v_a(0,0)\pi r}{4R_p} \tag{2.88}$$

If the actuator disk is combined with a nacelle, the term $v_a(0,0)$ is non-existing. It is therefore assumed that $v_a(0,0) = \overline{v_a(r,0)} = \int_{r_{hub}}^{r_{tip}} \frac{v_a(r,0)}{r_{tip}-r_{hub}} dr$.

Modification of the equations obtained by Conway

In order to be able to calculate the axial velocity at the outflow plane of the actuator disk, one has to know the radial distribution of the thrust. It is valid to assume that the velocity inside the ultimate slipstream is uniform, implying that the induced axial velocity will also have a uniform distribution. This assumption enables the use of expressions for a uniform induced axial velocity as obtained by Phillips (Phillips [2004]). A short derivation of these equations is given below. The elemental thrust of an annular element, $dT(r)$, can be found by considering the momentum equation in axial direction inside the ultimate slipstream.

$$dT(r) + (p_\infty - p_s)dA_s = (V_{a_s} - V_\infty)d\dot{m} \tag{2.89}$$

with:
$d\dot{m} = \rho V_{a_s} dA_s$
$dA_s = 2\pi r_s dr_s$

The term V_{a_s} is the axial velocity in the ultimate slipstream and is given by Eq (2.90) in which the term $v_a(r,\infty)$ is assumed to be uniformly distributed.

$$\begin{aligned} V_{a_s} &= V_\infty + v_a(r,\infty) \\ &= V_\infty + \overline{v_a(r,\infty)} \\ &= V_\infty + \int_{r_{hub}}^{r_{tip}} \frac{v_a(r,\infty)}{r_{tip}-r_{hub}} dr \\ &= V_\infty + 2v_i \end{aligned} \tag{2.90}$$

(2.91)

The term v_i in Eq (2.91) denotes the axial induced velocity of the uniform theory for which $v_a(r,0) = v_i$ should be true at any r. Note that with this theory v_i is independent of r and this is also the major shortcoming of this theory, since it implies that the load is uniformly distributed on the actuator plane ($dT(r) = constant$), which is usually not the case. Rewriting Eq (2.89) gives the following expression for dT:

$$dT(r) = 2\pi\rho r_s(V_{a_s} - V_\infty)V_{a_s}dr_s + (p_s - p_\infty)2\pi r_s dr_s \tag{2.92}$$

In order to find an expression for the pressure in the ultimate wake p_s, consider the continuity and momentum equation in radial coordinates. Since in the ultimate wake,

the considered axisymmetric flow becomes independent of the axial position, they take the following form:

$$\frac{\partial(r_s V_{r_s})}{\partial dr_s} = 0 \tag{2.93}$$

$$\rho\left(V_{r_s}\frac{\partial V_{r_s}}{\partial dr_s} - \frac{V_{\theta_s}^2}{r_s}\right) = -\frac{\partial p_s}{\partial r_s} \tag{2.94}$$

Since there is no radial velocity at the outer edge of the ultimate slipstream, Eq (2.93) reduces to:

$$V_{r_s} = 0 \tag{2.95}$$

Formula (2.93) states that there is no slipstream contraction in the ultimate wake.

Eq (2.95) then becomes:

$$\frac{\partial p_s}{\partial r_s} = \rho\frac{V_{\theta_s}^2}{r_s} = \rho\omega_s^2 r_s \tag{2.96}$$

The pressure at the outer edge of the slipstream, where $r_s = R_s$ is equal to the freestream pressure, p_∞. Using this boundary condition and integrating Eq (2.96) gives the following relation:

$$p_s = p_\infty - \frac{\rho\omega_s^2}{2}\left(R_s^2 - r_s^2\right) \quad \text{or} \quad p_s - p_\infty = -\frac{\rho\omega_s^2}{2}\left(R_s^2 - r_s^2\right) \tag{2.97}$$

The expressions for the induced angular and axial velocity in the ultimate slipstream are given by Eq (2.98) and Eq (2.99), respectively.

$$\omega_s = \frac{4\left(V_\infty + 2v_i\right)v_i}{\Omega_p R_p^2} \tag{2.98}$$

$$v_i = \sqrt{\frac{V_\infty^2}{4} + \frac{\Omega_p^2 R_p^2}{4}\left(1 - \sqrt{1 - \frac{4T}{A_p\rho\Omega_p^2 R_p^2}}\right)} - \frac{V_\infty}{2} \tag{2.99}$$

The expressions for the terms r_s, dr_s and R_s can be found by considering the continuity equation:

$$\pi r_p^2 \rho\left(V_\infty + v_i\right) = \pi r_s^2 \rho V_{a_s} \tag{2.100}$$

Rewriting Eq (2.100) in terms of r_s leads to the following expression for r_s:

$$r_s = \sqrt{\frac{V_\infty + v_i}{V_{a_s}}}\, r_p \tag{2.101}$$

The formulae for R_s and dr_s are given by Eq (2.102) and Eq (2.103) respectively:

$$R_s = \sqrt{\frac{V_\infty + v_i}{V_{a_s}}}\, R_p \tag{2.102}$$

$$dr_s = \sqrt{\frac{V_\infty + v_i}{V_{a_s}}} dr \tag{2.103}$$

where the term dr is the elemental width of an annular element at the outflow plane of the actuator disk.

Now that all necessary parameters in the ultimate uniform slipstream are known, one is able to calculate the elemental thrust term $dT(r)$ by means of Eq (2.92). The obtained elemental thrust can now be propagated back into Eq (2.87) to yield a non-uniform distribution of $v_a(r,0)$. This relieves the uniform distribution hypothesis at the actuator plane. In this newly obtained equation, the term $v_\theta(r,\infty)$ denotes the tangential induced velocity in the ultimate slipstream and can also be expressed as follows:

$$v_\theta(r,\infty) = \omega_s r_s \tag{2.104}$$

Where the term ω_s stands for the angular induced velocity in the ultimate slipstream as given by Eq (2.98).

By filling in the expression for ω_s into Eq (2.87) and rewriting in terms of $v_a(r,0)$ the following ordinary second order equation is found:

$$v_a^2(r,0) + V_\infty v_a(r,0) - \left(\frac{v_\theta^2(r,\infty)}{4} + \frac{dT(r)}{4\pi\rho r dr} \right) = 0 \tag{2.105}$$

Solving for $v_a(r,0)$ leads to the following expression for the axial induced velocity:

$$v_a(r,0) = \frac{1}{2}\left(-V_\infty + \sqrt{V_\infty^2 + v_\theta^2(r,\infty) + \frac{dT(r)}{\rho\pi r dr}} \right) \tag{2.106}$$

since the term $\frac{1}{2}\left(-V_\infty - \sqrt{V_\infty^2 + v_\theta^2(r,\infty) + \frac{dT(r)}{\rho\pi r dr}} \right)$ has no physical meaning.

The axial velocity at the outflow plane of the actuator disk is given by Eq (2.107).

$$V_a(r,0) = V_\infty + v_a(r,0) \tag{2.107}$$

By filling in the expression for v_a into Eq (2.107), formula (2.108) is obtained.

$$V_a(r,0) = \frac{V_\infty}{2} + \frac{1}{2}\left(\sqrt{V_\infty^2 + v_\theta^2(r,\infty) + \frac{dT(r)}{\rho\pi r dr}} \right) \tag{2.108}$$

Since the tangential velocity is zero before the actuator disk, the tangential velocity can be obtained by combining Eq (2.84), Eq (2.85) and Eq (2.86) which leads to Eq (2.109).

$$V_\theta(r,0) = \frac{4\pi v_a(r,0)\left(V_\infty + v_a(r,0)\right)}{2\pi r \Omega_p} \tag{2.109}$$

The radial velocity is also zero before the actuator disk. The radial velocity at the actuator plane can then be obtained by using Eq (2.110):

$$V_r(r,0) = v_r(r,0) = -\frac{v_a(0,0)\pi r}{4R_p} \tag{2.110}$$

Eq (2.108), Eq (2.109) and Eq (2.110) will be imposed at the actuator disk plane. The rest of the computational domain, including the slipstream and its contraction initiated at the actuator plane will be modelled using the Navier-Stokes equations.

Chapter 3

Numerical considerations

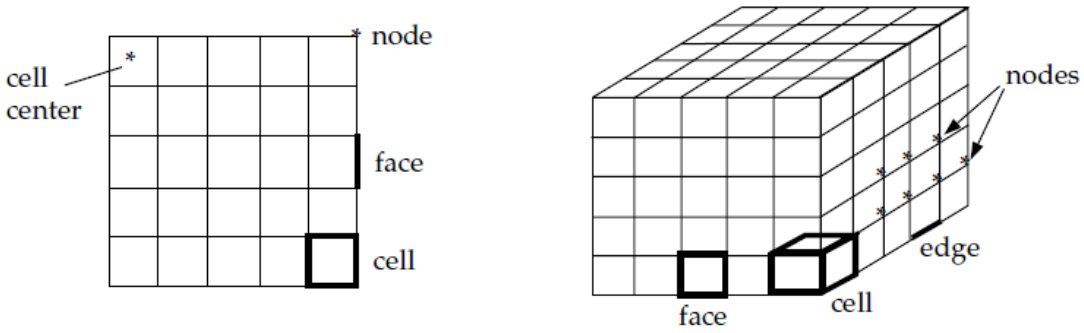
Chapter 2.3.3 discussed the derivation of the mathematical model which will be used in the propeller-wing interaction study. As stated before, in order to ensure that the model will perform its job correctly, it has to be validated first. This means that the results of the chosen mathematical model have to agree well with reference data. For the validation process, the wing will be omitted and a simulation will be performed which only involves a propeller-nacelle configuration. In order to perform the simulation, the mathematical model has to be translated into a UDF and a suitable mesh needs to be created. It is inevitable that the UDF will depend on the geometric features of the propeller-nacelle configuration and hence its mesh. It is therefore recommended that the reader is familiar with the basic mesh terminology and procedure in order to understand the UDF set-up which will be treated in Chapter 4.2. Therefore the first section of this chapter is dedicated to the basic mesh terminology and general guidelines for meshing. The second section will treat the FLUENT settings which were used during the validation and/or propeller-wing interaction study.

3.1 General meshing terminology and guidelines

A mesh consists of different control volumes or cells. Each cell is defined by a set of nodes, a cell center (centroid) and the faces that bound the cell. The terminology for a cell, face, node and cell center can be found in Figure 3.1 for 2D and 3D grids.

The different cell types that can be used in 2 or 3-dimensional grids is depicted in Figure 3.2. In a 2-dimensional grid, there are 2 different cell types: quadrilateral and triangular. Hexahedral, tetrahedral, pyramid, wedge, and polyhedral cells are 3-dimensional cell types.

The choice of cell type is important since it affects the time spent on grid creation, the computational cost and the amount of numerical diffusion. A grid can be structured or unstructured. In most cases, quadrilateral and hexahedral cells are used in structured grids, while unstructured grids consist of triangular or tetrahedral cells. However it is also



(a) Terminology for a 2D grid

(b) Terminology for a 3D grid

Figure 3.1: Terminology for a 2D (left) and 3D (right) cell "ANSYS FLUENT 12.0 UDF Manual" [2009]

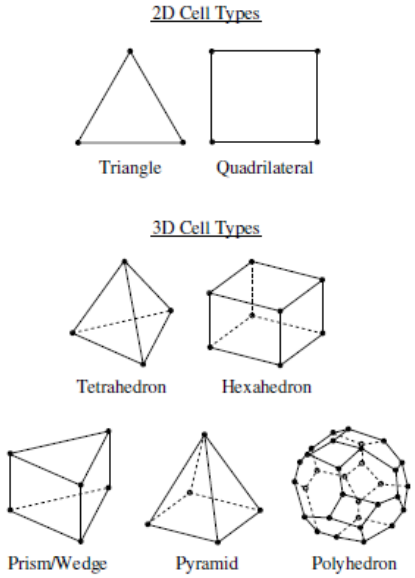


Figure 3.2: Cell types accepted by Fluent "ANSYS FLUENT 12.0 User's Guide" [2009]

possible to create unstructured grids with quadrilateral or hexahedral cells. For complex geometries, the creation of a structured grid is more time-consuming than the setup of an unstructured grid. Depending on the grid geometry, triangular or tetrahedral cells will fill up the grid with less cells than using their quadrilateral or hexahedral counterparts and vice-versa.
The skewness of a cell also determines the type of cell that should be used. The skewness of a cell depends on its aspect ratio. Skewness is defined as the difference between the shape of the cell and the shape of an equilateral cell of equivalent volume. A large aspect ratio (a measure of the stretching of a cell) in a triangular or tetrahedral cell can result in a high skewness factor, which affects the accuracy and convergence of the solution.
Numerical diffusion is minimized when the flow is aligned with the mesh. In triangular

or tetrahedral cells this is never the case. For simple flows, it could be the case for quadrilateral or hexahedral cells, however in complex flows alignment of the flow with the mesh will probably not occur either.
Another important consideration in grid generation is the change in cell volume between adjacent cells. If the cell volume between adjacent cells changes rapidly, large truncation errors will appear. The truncation error is the difference between the partial derivatives in the governing equations and their discrete approximations. So minimizing the truncation error minimizes the error introduced by discretizing the governing equations, which will lead to more accurate results.

The above stated guidelines were followed as much as possible for the mesh creation of the validation model and the propeller-wing interaction study. For a detailed discussion about the geometry and meshing procedure of the validation simulations, the reader is referred to section 4.1.

3.2 General FLUENT settings

The validation and propeller-wing interaction simulations have been performed with the commercial software FLUENT and more particularly the FLUENT 12.0.16 version was used. This chapter will elaborate on the settings that can be applied with FLUENT and is based on references "ANSYS FLUENT 12.0 User's Guide" [2009], "ANSYS FLUENT 12.0 Theory Guide" [2009], "ANSYS FLUENT 12.0 UDF Manual" [2009], Davidson [2004] and Marinus [2009]. Since there is such a vast availability of FLUENT settings, only a detailed discussion of the FLUENT settings used in the validation process and the propeller-wing interaction study will be presented here. For a detailed discussion of all the other FLUENT settings, the reader is referred to the FLUENT manuals "ANSYS FLUENT 12.0 User's Guide" [2009] and "ANSYS FLUENT 12.0 Theory Guide" [2009]. If the reader is already familiar with FLUENT, he is allowed to skip this section.

The main FLUENT settings can be split into the following 4 main categories:

1. Model settings
2. Material properties
3. Solver controls
4. Boundary conditions

The model settings are used to characterize the flow, e.g. whether it is steady or unsteady, laminar or turbulent, (in)compressible. FLUENT provides the option to define the material properties, not only of the solid used in the geometric model (aluminum nacelle for example), but also the properties of the fluid itself can be adjusted. The solver controls are used to define the used algorithm and discretization of the governing equations in order to model the flow numerically (e.g. first order upwind or second order scheme). The boundary conditions are used to define the type of faces and volumes of the imported mesh (e.g. wall, pressure inlet).

3.2.1 Model settings

The model settings specify the type of flow. The main specifications involve the space (2D or 3D flow), time ((un)steady), viscous settings and flow state (laminar or turbulent). Other additional settings involve heat transfer, radiation, pollutants and species transport. The choice of the turbulence model depends on the type of flow, the required accuracy level and the available computational resources.
Turbulence is a result of the instability of the mean flow that causes large vortical structures (eddies) to break-up or evolve into smaller vortices. For flows with a high Reynolds number, there is a broad spectrum of eddy sizes that appear in the flow. According to Richard's Energy Cascade theory this breaking-up of the vortices into smaller and smaller vortices continues until the Reynolds number based on the length scale of the smallest eddy reaches unity. Figure 3.3 shows a schematic representation of the break-up of the large vortices into smaller eddies according to the energy cascade concept.

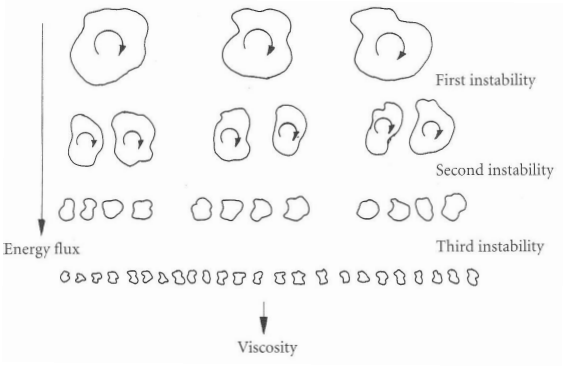


Figure 3.3: Richard's energy cascade. "Davidson [2004]"

In theory it is possible to resolve (compute) all the turbulent length scales by applying direct numerical simulation (DNS) since turbulent flows are fully described by the Navier-Stokes Equations. However DNS is not an option for high Reynolds flow (break-up into smaller and smaller length scales) since the computational cost is proportional to Re^3. Instead of computing all turbulent length scales, one could only resolve the large eddies directly in a time-dependent simulation while the small eddies are modelled. This approach is called Large Eddy Simulation (LES) and uses less computational resources than DNS but is not suitable for most practical applications because of its high computational cost.

Turbulent flows are characterized by the following two properties:

1. The velocity field fluctuates randomly in time, is highly disordered in space and contains a wide range of length scales.

2. The velocity field is unpredictable in the sense that a small change to the initial conditions will produce a large change to the subsequent motion.

Although the velocity field, $u(x,t)$ is random and unpredictable, its statistical properties are not. So by averaging the Navier-Stokes equations, the statistical properties of the

flow can be determined. This procedure in which one statistically averages the Navier-Stokes equations before solving them is very common in today's practices because of its low computational cost. But what is meant by the term "averaging"? The most common approach is to time-average the Navier-Stokes equations, which leads to the Reynolds-Averaged Navier-Stokes equations or RANS. This is done by applying Reynolds and Favre averaging which will now shortly be illustrated.
Reynolds averaging an instantaneous velocity component e.g. $u(x,t)$ is done by specifying this variable as the sum of its mean value, $\bar{u}$ and its fluctuating component $u'(x,t)$ as can be seen from Eq (3.1).

$$u(x,t) = \bar{u} + u'(x,t) \tag{3.1}$$

The time-averaged values of the velocity is calculated as follows:

$$\bar{u} = \int_{-\frac{T}{2}}^{+\frac{T}{2}} u(t+\tau)d\tau \tag{3.2}$$

Favre averaging on the other hand is performed as follows:

$$u_i(x,t) = \tilde{u}(x,t) + u''(x,t) \tag{3.3}$$

with

$$\tilde{u}(x,t) = \frac{1}{\bar{\rho}T}\int_{-\frac{T}{2}}^{+\frac{T}{2}} \rho(x,t)u(x,t)dt \tag{3.4}$$

In order to obtain the RANS equations, the density and pressure terms are Reynolds-averaged, while all the other terms are Favre-averaged. The resulting Reynolds-Averaged equations are given by Eq (3.5) to Eq (3.7).

$$\frac{\partial\bar{\rho}}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_i}{\partial x_i} = 0 \tag{3.5}$$

$$\frac{\partial\bar{\rho}\tilde{u}_i}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_i\tilde{u}_j}{\partial x_j} = -\frac{\partial\overline{\rho u_i'' u_j''}}{\partial x_j} - \frac{\partial\bar{p}}{\partial x_i} + \frac{\partial\overline{2\mu(S_{ij} - 1/3\delta_{ij}S_{ii})}}{\partial x_j} \tag{3.6}$$

$$\frac{\partial\bar{\rho}\tilde{E}}{\partial t} + \frac{\partial\bar{\rho}\tilde{u}_j\tilde{H}}{\partial x_j} = \frac{\partial}{\partial x_j}\left[-\bar{q}_j - \overline{\rho u_j'' h''} + \overline{\tau_{ij}'' u_i''} - \overline{1/2\rho u_j'' u_i'' u_i''}\right] \tag{3.7}$$
$$+\frac{\partial}{\partial x_j}\left[\tilde{u}_i(\bar{\tau}_{ij} - \overline{\rho u_i'' u_j''})\right]$$

By inspecting the RANS equations it can be seen that there are two additional terms, which are a direct result of averaging. The Reynolds-stress term $-\overline{\rho u_i'' u_j''}$ and the turbulent heat flux vector $\overline{\rho u_j'' h''} - \overline{\tau_{ij}'' u_i''} + \overline{1/2\rho u_j'' u_i'' u_i''}$. These additional terms are responsible for the so called "closure problem" since the Reynolds-stress term creates 6 additional unknowns in the equations while the turbulent heat flux vector introduces 3 additional unknowns. Over time several "closure models" have been proposed in order to solve the RANS equations.
One of these models is the "Eddy viscosity assumption". This assumption results from

Boussinesq hypothesis which relies on a turbulent viscosity μ_t, depending on flow features, to connect the Reynolds-stress components to the strain rate. This hypothesis is quite similar to Newton's law for molecular viscosity. This hypothesis can be written as:

$$\tau_{ij}^{Reynolds} = -\overline{\rho u_i'' u_j''} = 2\mu_t(\tilde{S}_{ij} - 1/3\delta_{ij}\tilde{S}_{kk}) - 2/3\rho k\delta_{ij} \tag{3.8}$$

For more information concerning the other closure models the reader is referred to one of the references as stated above.
Since the validation and propeller-wing interaction simulations make use of the same turbulence model (Realizable k-ε) only this model will be shortly discussed. But a short introduction to the law of the wall will be given first.

The law of the wall

The presence of a boundary (wall) has a profound effect on a turbulent shear flow. Not only does the flow need to comply with the no-slip condition, also the physical behaviour of the flow is altered due to the presence of the wall which reduces the normal and tangential velocity fluctuations very close to the wall. One can distinct 3 layers in a turbulent flow near the wall: the viscous sublayer, the buffer layer and fully-turbulent layer, see Figure 3.4.

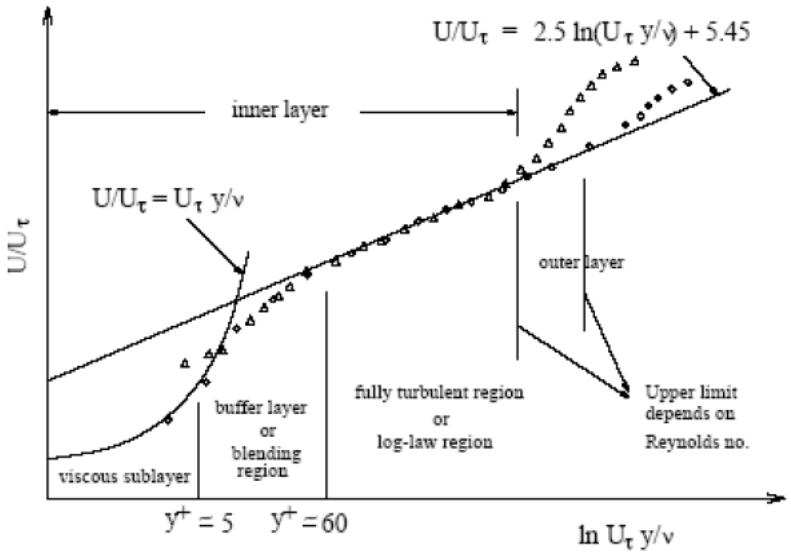


Figure 3.4: Different layers in a turbulent flow close to the wall. "Marinus [2009]"

The viscous sublayer is that part of the flow that is the closest to the wall. The flow is there almost laminar and the molecular viscosity plays a dominant role in the momentum, heat and mass transfer. The fully-turbulent layer is the most outer layer and is dominated by turbulence. The buffer layer is situated in between the viscous sublayer and fully-turbulent layer. Here, the effects of molecular viscosity and turbulence are equally important.

Models based on the eddy viscosity concept are constructed to be consistent with the law of the wall i.e. the expression of the velocity profile in the logarithmic part of the

boundary layer (See Figure 3.5). One defines $u_\tau = \sqrt{\tau_w/rho}$, $u^+ = u/u_\tau$ and $y^+ = yu_\tau/\nu$. Derivation of the Navier-Stokes equations in the boundary layer close to the wall, where the viscous stresses are dominant, leads to Eq (3.9)

$$u^+ = y^+ \tag{3.9}$$

In this viscous sub-layer, one can show that the dissipation at the wall is non-zero and a limiting value for ε can be computed: $\varepsilon = 2\nu k/y^2$ for $y \to 0$ or in terms of $\omega = \varepsilon/(c_\mu k) = 2\nu/(c_\mu y^2)$.

Furthermore, if one admits that close to the wall but not extremely close, there is a region where the velocity profile is determined by the turbulent shear stress but not by the shear stress due to molecular viscosity, one can deduce the following relation:

$$u^+ = \frac{1}{\kappa} \ln y^+ + B \tag{3.10}$$

where κ is the von Karman constant and B another constant. This law is valid in the logarithmic layer and expresses that in this layer, the Reynolds stresses are dominant on viscous stresses. Different experimental results tend to confirm this hypothesis and the validity of the law of the wall (logarithmic profile). Generally, the law of the wall applies from $y^+ = 30$ to $y/\delta \approx 0.1$.

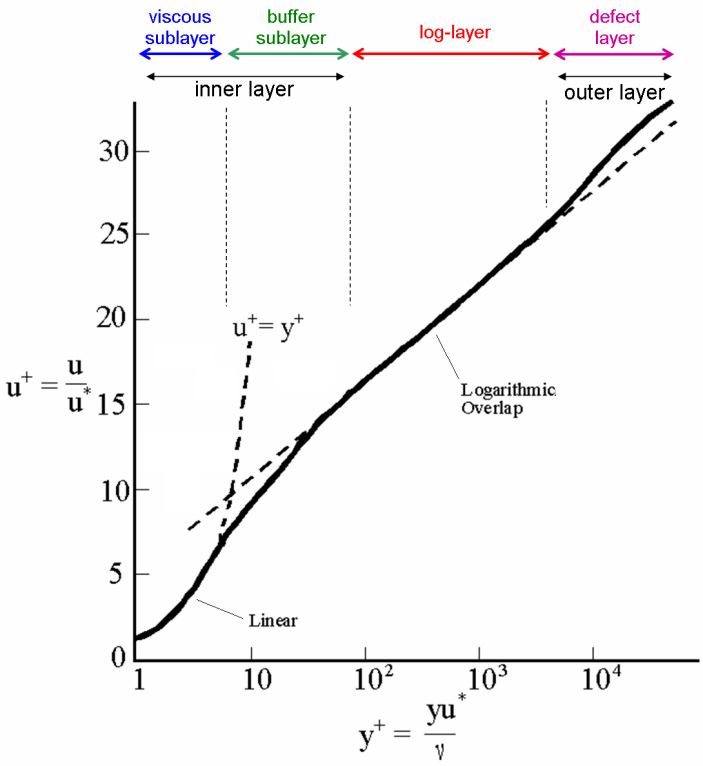


Figure 3.5: Velocity profile in a turbulent boundary layer. "Marinus [2009]"

Standard $k-\varepsilon$ model

The standard $k-\varepsilon$ model was developed by Jones and Launder in 1972 and uses the RANS-equations together with the TKE-equation (Eq(.3.11)), the ε-equation (Eq(3.12)), and the eddy-viscosity (Eq (3.13)).

$$\frac{D\bar{\rho}k}{Dt} = \bar{\rho}P_k - \bar{\rho}\varepsilon + \frac{\partial}{\partial x_i}\left[(\mu + \frac{\mu_t}{\sigma_k})\frac{\partial k}{\partial x_i}\right] \tag{3.11}$$

$$\frac{D\bar{\rho}\varepsilon}{Dt} = c_{\varepsilon 1}\bar{\rho}\frac{\varepsilon}{k}P_k - c_{\varepsilon 2}\bar{\rho}\frac{\varepsilon}{k}\varepsilon + \frac{\partial}{\partial x_i}\left[(\mu + \frac{\mu_t}{\sigma_\varepsilon})\frac{\partial \varepsilon}{\partial x_i}\right] \tag{3.12}$$

$$\nu_t = c_\mu \frac{k^2}{\varepsilon} \tag{3.13}$$

where σ_k is the Prandtl number and $P_k = 2\nu_t \tilde{S}_{ij}\tilde{S}_{ij}$.

Standard values for the constants c_μ, σ_k, σ_ε, $c_{\varepsilon 1}$ and $c_{\varepsilon 2}$ have been obtained from simple flows. The constant c_μ comes from the observation of thin shear flows with approximate balance between production and dissipation, the constant $c_{\varepsilon 2}$ from the decay of homogeneous turbulence and the constant $c_{\varepsilon 1}$ from homogeneous shear flow experiments. The Prandtl number σ_k is a priori taken as unity since it governs the turbulent diffusion of the TKE by the turbulent motion itself. And the Prandtl number σ_ε comes from consideration of the diffusion of ε in the logarithmic layer. For that Prandtl number, it is imperative to satisfy Eq (3.14) in order to ensure that the law of the wall can be predicted correctly.

$$\sigma_\varepsilon = \frac{\kappa^2}{(c_{\varepsilon 2} - c_{\varepsilon 1})\sqrt{c_\mu}} \tag{3.14}$$

The set of equations for this model is:

$$\begin{aligned} \frac{Dk}{Dt} &= P_k - \beta^* \omega k + \frac{\partial}{\partial x_i}\left[(\nu + \sigma^* \nu_t)\frac{\partial k}{\partial x_i}\right] \\ \frac{D\bar{\rho}\varepsilon}{Dt} &= c_{\varepsilon 1}\bar{\rho}\frac{\varepsilon}{k}P_k - c_{\varepsilon 2}\bar{\rho}\frac{\varepsilon}{k}\varepsilon + \frac{\partial}{\partial x_i}\left[(\mu + \frac{\mu_t}{\sigma_\varepsilon})\frac{\partial \varepsilon}{\partial x_i}\right] \end{aligned} \tag{3.15}$$

Realizable $k-\varepsilon$ model

The realizable $k-\varepsilon$ model differs from the standard $k-\varepsilon$ model in:

- A critical coefficient of the model, c_μ, is expressed as a function of mean flow and turbulence properties, rather than assumed to be constant as in the standard model.
- A new expression for the dissipation rate.

This allows the model to satisfy certain mathematical constraints on the normal stresses consistent with the physics of turbulence (realizability). The realizable $k-\varepsilon$ model has shown substantial improvements over the standard $k-\varepsilon$ model where the flow features include strong streamline curvature, vortices and rotation.

Wall functions

When low-Reynolds modelling is used (i.e. when the k-ε or k-ω models are used up to the wall), the near-wall behaviour of turbulent quantities must be carefully modelled. Indeed, close to the wall, turbulence is damped and the turbulent Reynolds number $Re_t = k^2/(\nu\varepsilon)$ becomes small. To achieve this, damping functions are introduced into Equation (3.15). In many applications, the precise description near the walls is not needed and high Reynolds modelling can be used. As with low Reynolds modelling the first grid point should be at $y^+ \approx 1$, the numerical cost is elevated because around 50 grid points are necessary in the boundary-layer. In high Reynolds modelling, the first grid point is typically so that $y^+ \in [30, 100]$. At this point P, special boundary conditions are imposed which are given by Eq (3.16):

$$\tau_P = \tau_w \qquad k_P = \frac{u_\tau^2}{\sqrt{c_\mu}} \qquad \varepsilon_P = \frac{u_\tau^3}{\kappa y} \tag{3.16}$$

where u_τ is determined from

$$\frac{u}{u_\tau} = \frac{1}{\kappa}\ln\left(\frac{B_2 y u_\tau}{\nu}\right)$$

with u from the previous iteration, $\kappa = 0.41$ and $B_2 = 7.768$. The backdraw of this method is that it corresponds to imposing a constant value of k at the first grid point in the logarithmic-layer although in reality, the TKE has a pronounced maximum in the buffer-layer so there is no zone of constant TKE between the first grid point and the wall. The TKE-equation (Eq(3.11)) can be solved at the first grid point by imposing $P_k = \tau_P \partial u/\partial y$ with $\partial u/\partial y = u_\tau/(\kappa y)$. Then ε can be determined as $u_\tau = k^{1/2} c_\mu^{1/4}$ so $\varepsilon = c_\mu^{3/4}/(\kappa y)$. In this, u_τ or τ_w are determined from the following formulae:

$$\begin{aligned} &\frac{u}{u_\tau}\frac{u_\tau}{u_\tau} = \frac{1}{\kappa}\ln\left(\frac{B_2 y u_\tau}{\nu}\right) \Rightarrow \frac{u\sqrt{k}c_\mu^{1/4}}{u_\tau^2} = \frac{1}{\kappa}\ln\left(B_2 y\frac{\sqrt{k}c_\mu^{1/4}}{\nu}\right) \\ &\Rightarrow u_\tau^2 = \frac{\tau_w}{\rho} = \frac{\sqrt{k}c_\mu^{1/4}\kappa}{\ln(B_2 y_P^+)}u \qquad \text{where } y_P^+ = \frac{c_\mu^{1/4}\sqrt{k}y_P}{\nu} \end{aligned} \tag{3.17}$$

In this approach, the assumption $\mu = \mu_t$ is made so that molecular viscosity is set to zero. The high Reynolds approach is not valid for flows with re-circulation as for these flows, a sign-change occurs for u_τ. $|u_\tau|$ is used in the determination of y^+ but the condition $y^+ \in [30, 100]$ cannot be satisfied in the recirculation region (generally with lower velocities). Although the above algorithm does not cause problems for regions of recirculation (because u_τ is replaced by $c_\mu^{1/4}\sqrt{k}$ for the computation of y^+), blindly applying it corresponds to enforcing the log-law in a region where the laws of the viscous sub-layer should be used. In such a case, standard wall functions are inappropriate and are often replaced by a two-layer approach so that low Reynolds modelling does not have to be implemented. This approach provides two sets of equations for which the choice of the to be used equations depends on the value of y_P^+:

$$y_P^+ = \frac{c_\mu^{1/4}\sqrt{k}y_P}{\nu}$$

$y_P^+ > 11.3 \rightarrow$ log-law	$y_P^+ \leq 11.3 \rightarrow$ linear-law
$\tau_w = \frac{\rho\sqrt{k_P}c_\mu^{1/4}\kappa}{\ln(B_2 y_P^+)}u_P$	$\tau_w = \frac{\mu}{y_P}u_P$
$u_\tau = \sqrt{\frac{\lvert\tau_w\rvert}{\rho}}\left(\frac{\tau_w}{\lvert\tau_w\rvert}\right)$	$u_\tau = \sqrt{\frac{\lvert\tau_w\rvert}{\rho}}\left(\frac{\tau_w}{\lvert\tau_w\rvert}\right)$
$u_y = \frac{u_\tau}{\kappa y}$	$u_y = \frac{\tau_w}{\mu}$
$P_k = \tau_w u_y$	$P_k = \tau_w u_y$
$\varepsilon = \frac{c_\mu^{3/4}k^{3/2}}{\kappa y}$	$\varepsilon = \frac{2\nu k_P}{y_P^2}$

3.2.2 Material properties

The material properties of the fluids and solids used in the simulation can be defined by the user. The fluid properties that can be adjusted are the density (e.g. constant, ideal gas), specific heat capacity c_p (e.g. constant, piecewise linear), thermal conductivity, viscosity(e.g. constant, Sutherland) and molecular weight. For a solid the density, c_p and thermal conductivity need to be defined.

3.2.3 Solver controls

There are two different solvers available in FLUENT:

1. Pressure-based solver
2. Density-based solver

Both solvers use a finite-volume discretization to discretize the governing flow equations, however the approach to linearize and solve the obtained discretized equations is different. The velocity field of the solution is by both methods obtained from the momentum equations. However the pressure field is determined differently. The density-based solver determines the pressure field by means of the Equation Of State. On the other hand the pressure-based solver calculates the pressure field by solving a pressure or pressure correction equation, which is obtained by manipulating the continuity and momentum equations in such a way that the velocity field, which is corrected by the pressure equation, satisfies the continuity equation. For both algorithms, the governing equations are non-linear and coupled. However the solution process of these coupled equations are different. The density-based approach solves the continuity, momentum, energy and species equations simultaneous while the pressure-based does not. The solution procedure of the pressure-based solver decouples the governing equations and solves them sequentially. There are however two ways of doing this, either the Pressure-based Segregated algorithm or the Pressure-based Coupled algorithm is applied. As the name segregated implies, each governing equation is decoupled from the other equations during the solution process and are solved one after another. The Pressure-based Coupled algorithm on the other hand sees the momentum and pressure-based continuity equation as a coupled, non-segregatable system. The remaining equations are however again decoupled and solved sequentially as

was the case in the Segregated algorithm. The difference between the two Pressure-based and the Density-based algorithm is depicted in Figure 3.6. The integral form of the unsteady conservation equation for the transport scalar, ϕ, for an arbitrary control volume Ω is given by formula (3.18)

$$\int_{\Omega} \frac{\partial \rho \phi}{\partial t} d\Omega + \oint \rho \phi \vec{V} d\vec{A} = \oint \Gamma_{\phi} \nabla \phi d\vec{A} + \int_{\Omega} S_{\phi} d\Omega \tag{3.18}$$

Where:
ρ = density
$\vec{V}$ = velocity vector
$\vec{A}$ = surface area vector
Γ_{ϕ} = diffusion coefficient of ϕ
$\nabla\phi$ = gradient of ϕ
S_{ϕ} = source of ϕ per unit volume

Discretizing Eq (3.18) by means of the finite volume method leads to formula (3.19).

$$\frac{\partial \rho \phi}{\partial t} \Omega + \sum_{f}^{N_{faces}} \rho \vec{V_f} \phi_f \vec{A_f} = \sum_{f}^{N_{faces}} \Gamma_{\phi} \nabla \phi_f \vec{A_f} + S_{\phi} \Omega \tag{3.19}$$

Where:
N_f = number of faces enclosing the cell
ϕ_f = value of ϕ convected through face f
$\rho \vec{V_f} \phi_f \vec{A_f}$ = mass flux through face f
$\vec{A_f}$ = area of face f
$\nabla\phi_f$ = gradient of ϕ at face f
Ω = cell volume

FLUENT stores the discrete values of the scalar quantity ϕ at the cell centers, however inspection of Eq (3.19) shows that the face values need to be known as well. There are 8 different discretization schemes in FLUENT available for the discretization of the convection terms of Eq (3.19) in order to obtain the face values of the scalar quantity ϕ. They consist of the first and second-order Upwind Scheme, the Power-Law Scheme, Central and Bounded Differencing Scheme, Quick Scheme, third-order MUSCL Scheme and the modified HRIC Scheme. The choice of the discretization type depends on the complexity of the mesh and the accuracy needed. The Central and Bounded Differencing Scheme are only available when one uses the LES turbulence model while the QUICK Scheme works only for hexahedral meshes and works best when the cells are aligned with the flow. As with all numerical methods, each discretization method has it benefits and disadvantages.

The gradient term $\nabla\phi$ present in Eq (3.19) needs to be evaluated as well. FLUENT offers 3 different methods for gradient computation:

1. Green-Gauss Cell-Based

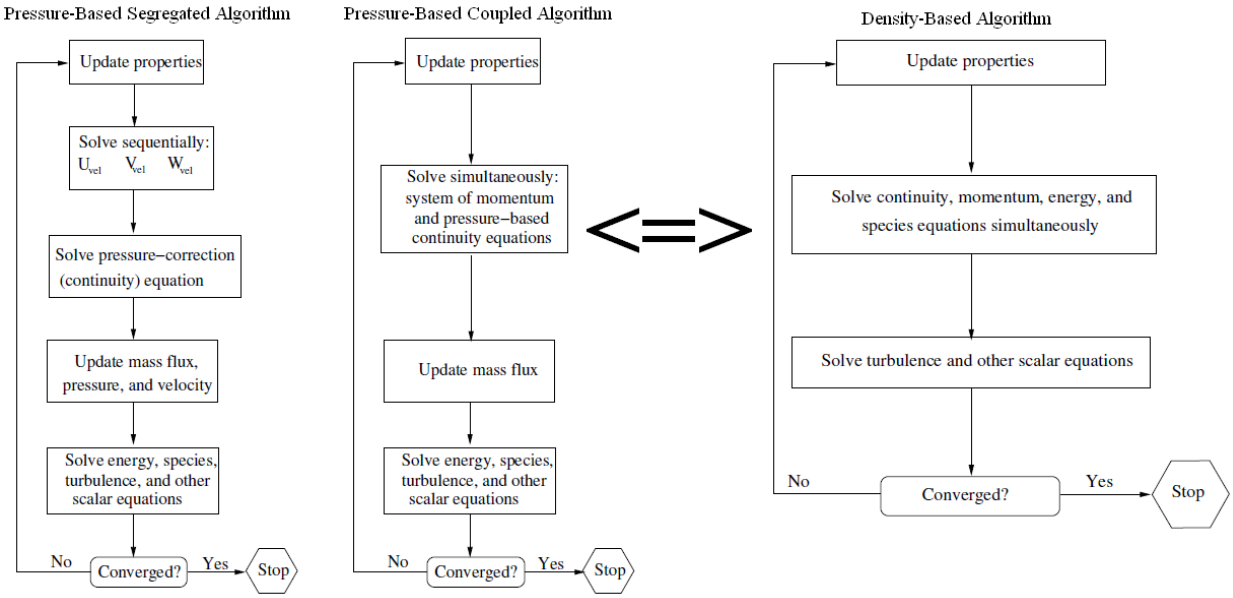


Figure 3.6: Difference in solution process of Pressure-Based and Density-Based Algorithms. "ANSYS FLUENT 12.0 Theory Guide" [2009]

2. Green-Gauss Node-Based
3. Least Squares Cell-Based

Eq (3.20) represents the Green-Gauss computation theorem in order to compute the gradient of the scalar entity ϕ at the cell center $c0$.

$$(\nabla\phi)_{c0} = \frac{1}{V}\sum_{f}\bar{\phi}_f\vec{A}_f \tag{3.20}$$

The difference in the Green-Gauss Cell-Based and Node-Based approach lies in the evaluation of the term $\bar{\phi}_f$. The Cell-Based Green-Gauss theorem takes the arithmetic mean of the values of the neighbouring cell centers to calculate the face value, ϕ_f:

$$\bar{\phi}_f = \frac{\phi_{c0} + \phi_{c1}}{2} \tag{3.21}$$

The Node-Based Green-Gauss theorem on the other hand takes the arithmetic mean of the nodal values of the face to calculate the face value, ϕ_f:

$$\bar{\phi}_f = \frac{1}{N_f}\sum_{n}^{N_f}\bar{\phi}_n \tag{3.22}$$

The term N_f appearing in Eq(3.22) represents the number of nodes on the face while the term $\bar{\phi}_n$ represent the nodal values which are obtained by calculating the weighted

average of the cell values surrounding the nodes. For unstructured meshes the node-based gradient evaluation is more accurate than the cell-based gradient calculation.

The Least-Squares Cell-Based Gradient theorem assumes a linear solution. Therefore one can express the the change in cell values between cell $c0$ and ci along the vector δr_i, connecting cell $c0$ and ci (See Figure 3.7) as follows:

$$(\nabla\phi)_{c0} \cdot \delta r_i = (\phi_{ci} - \phi_{c0}) \tag{3.23}$$

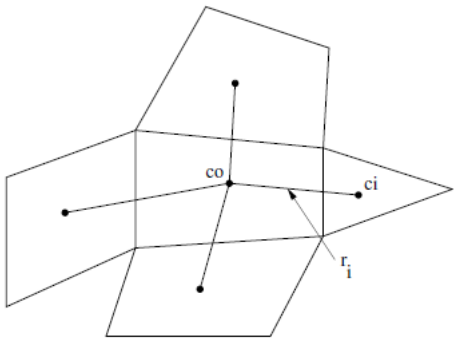


Figure 3.7: Least-Squares Gradient determination. "ANSYS FLUENT 12.0 Theory Guide" [2009]

This procedure can be repeated for all cells surrounding the cell $c0$, which leads to the following expression:

$$[J](\nabla\phi)_{c0} = \delta\phi \tag{3.24}$$

The cell gradient is then obtained by solving the minimization problem of Eq (3.24) in a least-squares sense.

By inspection of Eq (3.19), one can see that the values of the density at the faces needs to be known as well. For incompressible flows, FLUENT calculates the arithmetic average to obtain the density value at a face. For compressible flows however, this method is not suitable. As was the case with the convective terms FLUENT provides the following methods to calculate the face density for a compressible flows: first and second-order Upwind Scheme, the Power-Law Scheme, Central and Bounded Differencing Scheme, Quick Scheme and third-order MUSCL Scheme.

As stated before, the Pressure-based algorithm discretizes the continuity and momentum equation and connects the pressure and velocity field by means of a pressure correction equation. This discretization of the continuity and momentum equation is similar to the discretization of Eq (3.18) if the pressure field and face mass fluxes are known. However this is not the case, therefore these values have to be obtained as a part of the solution. FLUENT uses a co-located scheme, in which pressure and velocity are stored at the cell center. In order to obtain the pressure values at the faces different pressure interpolation schemes can be used. The STANDARD scheme in FLUENT interpolates the pressure values at the faces by making use of momentum equation coefficients. As long as the pressure variation is smooth, this procedure works well. However when there are large jumps, this method is unsatisfactory. For these cases use can be made of the second-order scheme, the body-force-weighted average scheme or the PRESTO! scheme. The

PRESTO! (Pressure Staggering Option) is similar to the staggered-grid schemes and is useful for flows with high-swirl numbers, high-Rayleigh number natural convection and porous media while the body-force-weighted scheme is recommended for flows subjected to large body forces. The second-order scheme is best used for compressible flows and when the accuracy of the other schemes is insufficient.

For the Segregated Pressure-Based algorithm, different approaches to find the pressure correction equation can be used. The SIMPLE algorithm is FLUENT's default algorithm and is an iterative solution method. First the momentum equation is solved with a guessed pressure field p^*. The resulting face flux, J_f^* does not satisfy the continuity equation, therefore a correction factor, J_f' is added such that the continuity equation is satisfied leading to the following corrected face flux (J_f) equation:

$$J_f = J_f^* + J_f' \tag{3.25}$$

The SIMPLE algorithm assumes that the correction factor for the face flux can be written as Eq (3.26).

$$J_f' = d_f * \left(p_{c0}' - p_{c1}'\right) \tag{3.26}$$

The term p' represents the pressure correction for a cell. Eq (3.25) and (3.26) are substituted in the discrete continuity equation to obtain a discrete expression for the pressure correction term, p', which is expressed by Eq (3.27)

$$a_p p' = \sum_{nb} a_{nb}\, p_{nb}' + b \tag{3.27}$$

The term b is the net flow rate into the cell and is given by formula (3.28).

$$b = \sum_{f}^{N_{faces}} J_f^*\, A_f \tag{3.28}$$

The pressure-correction equation is solved using the AMG method (Algebraic Multigrid Method). After obtaining a solution for Eq (3.27), each cell pressure and face flux is corrected by means of respectively Eq (3.29) and Eq (3.30).

$$p = p^* + \alpha_p p' \tag{3.29}$$

$$J_f = J_f^* + d_f \left(p_{c0}' - p_{c1}'\right) \tag{3.30}$$

The term α_p in Eq (3.29) is the under-relaxation factor for the pressure.

A method similar to the SIMPLE algorithm is the SIMPLEC method (SIMPLE-Consistent). It follows the same procedure as the SIMPLE algorithm, except for the expression used for the face flux correction. The term d_f in this equation differs from the SIMPLE algorithm such that it accelerates convergence. For most problems, the SIMPLE and SIMPLEC algorithm is sufficient. However for transient flow calculations or calculations on highly skewed meshes, the PISO (Pressure-Implicit with Slitting Operation) scheme is recommended. This solution method is part of the SIMPLE family of algorithms but is based on a higher order of the approximate relation between the corrections for pressure and velocity. After one or more additional PISO iterations the corrected velocities satisfy the momentum equation better than the SIMPLE(C) algorithms. This decreases the number of iterations required for convergence with respect to the SIMPLE(C) method.

3.2.4 Boundary conditions

Boundary conditions need to applied to faces that confine a calculation domain. The following boundary conditions were used during the validation and propeller-wing interaction study:

1. Pressure Inlet
2. Pressure Outlet
3. Pressure Far-field
4. Mass flow Inlet
5. Wall
6. Periodic
7. Symmetry

3.2.5 Pressure in- and outlet

The difference in pressure of the pressure inlet and outlet boundaries is used to control the desired freestream Mach number. For the pressure inlet the so called "gauge" total pressure has to be specified. The gauge total pressure is defined as follows:

$$p_{gauge} = p_t - p_{op} \tag{3.31}$$

Thus the gauge pressure is the difference of the total and operating pressure. If the operating pressure is known, the gauge total pressure can be determined by using the following isentropic relation:

$$\frac{p_t}{p_{op}} = \left(1 + \frac{\gamma - 1}{2} M_\infty^2\right)^{\frac{\gamma}{\gamma - 1}} \tag{3.32}$$

Combining Eq (3.32) with Eq (3.31) leads to Eq (3.33).

$$p_{t_{gauge}} = p_{op}\left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}} - p_{op} \tag{3.33}$$

For the pressure outlet, the gauge static pressure has to be inputted. By setting the gauge static pressure of the pressure outlet boundary to zero, Eq (3.33) will give the correct gauge total pressure needed at the inlet to obtain the desired freestream Mach number. However, due to numerical losses, the gauge total pressure has to be set at a somewhat larger value than the calculated one.

Besides pressure, also the total temperature has to be specified for both the pressure in- as outlet(backflow total temperature). The total temperature can by determined by means of the isentropic relation given by formula (3.34).

$$\frac{T_t}{T_s} = \left(1 + \frac{\gamma - 1}{2} M_\infty^2\right) \tag{3.34}$$

Since no backflow is expected to occur, the backflow Mach number can be set to zero resulting in a total temperature for the outlet that is equal to freestream static temperature. Other important inputs for both the pressure in- and outlet boundary are turbulence parameters such as turbulent length scale and intensity.

Pressure Far-field

The "pressure far-field boundary condition" is used to model a freestream condition at infinity. Important input parameters are: gauge pressure, Mach number, static temperature, flow direction and turbulence parameters.

Mass flow Inlet

If a specific mass flow rate needs to be prescribed at a boundary, the mass flow inlet boundary condition is used. Required inputs are: the mass flow rate, total temperature, flow direction and turbulence parameters. By specifying the direction opposite to the flow direction, a mass flow outlet can be simulated as well.

Wall

If the "wall option " is chosen as boundary condition, the users has two options: either the wall is stationary or moving. Other important inputs are the shear options (no slip or specified shear) and the wall roughness parameters.

Symmetry

The symmetry boundary condition will only be used in the propeller-wing interaction study. By making use of this boundary condition, only half of the airplane needs to be modelled. This will reduce the amounts of cells needed and hence the calculation cost.

Periodic

The periodic boundary condition will only be used in the validation process and not in the propeller-wing interaction study. The geometry and flow pattern of the validation simulation are axis-symmetric. By making use of the rotational periodic boundary condition only a part of the model needs to be modelled. Hence the mesh will consist of fewer cells, leading to a reduction in computational cost.

Part II: Validation

This part of the thesis treats the validation of the numerical model that will be used in propeller-wing interaction study.

Chapter 4

Validation of the numerical propeller model

This chapter deals with the validation of the numerical model, which will be set up in section 4.2 and is based on the mathematical model developed in section 2.3.3. For the validation process, the wing will be omitted and a simulation will be performed which only involves a propeller-nacelle configuration. In order to evaluate the performance of the numerical model, it will be compared to the results obtained by full blade modelling using RANS (Marinus [2007]). To this end, flow parameters such as Mach number, static pressure, etc. are extracted along 3 different lines of constant radius ($50\%R_p$, $75\%R_p$ & $96\%R_p$) for both simulations and compared.
From the discussion in section 2.1.3 it followed that the presence of the nacelle changes the inflow velocity at the propeller blade. To investigate whether the inclusion of this change in inflow velocity will lead to more accurate results the non-uniform actuator disk model was created. This model makes also use of the same equations as the uniform actuator disk model, which assumes a uniform inflow velocity equal to freestream. The only difference between the equations of the uniform and non-uniform actuator disk model is the fact that the term V_∞ is replaced by the term V_{in}, the inflow velocity at the propeller blade.
The induced radial velocity is neglected in most numerical investigations concerning propeller-wing interaction effects. To see whether the inclusion of the induced radial velocity in the current numerical model delivers superior results with respect to the ones that neglect it, Phillips' equations (Phillips [2004]) are also implemented in a UDF. These equations neglect the induced radial velocity and assume a uniform inflow velocity equal to freestream.
This chapter contains 3 sections. The first section will treat the creation of the geometric model and its mesh. Then the setup of the UDF for the uniform and non-uniform actuator disk as well as Phillips' equations will be discussed. The third section will elaborate on the Fluent settings and why they were chosen as such. This chapter will conclude with a discussion of the performance of the different models and which one is best fit to investigate propeller-wing interference effects.

4.1 SR-1 mesh creation

The propeller that will be used in the validation simulations is the NASA SR-1 propeller because of the wide availability of data involving its shape. The SR-1 blade has a moderate sweep angle of 30° and was part of the NASA Advanced Turboprop Project (ATP). A scale model of this propeller was tested in the NASA Lewis Wind Tunnel. The scaled propeller has a diameter of 0.622 m and its characteristics can be found in table 4.1. For the validation process, the propeller will be modelled as an actuator disk and the nacelle will be included in the geometry. The coordinates specifying the spinner-nacelle geometry can be found in Appendix A.

Table 4.1: Installation characteristics of the SR-1 propeller

property	**value**
Diameter	0.622 m
x-coordinate LE of blade	0.13 m
x-coordinate TE of blade	0.215 m

The coordinate system has its origin at point (0,0,0), which corresponds to the nose of the spinner-nacelle geometry, the latter is axisymmetric around the x-axis. By importing the coordinates, which specify the spinner-nacelle geometry, in Gambit and drawing "NURBS" lines through the imported coordinates the nacelle geometry is obtained. NURBS stands for Non-Uniform Rational B-Spline and it is Gambit's curve-fitting routine, which fits a smooth curve onto several vertices. ("GAMBIT 2.4 Modeling Guide" [2007])

The x-axis is the axisymmetric symmetry axis and the x-velocity corresponds to the axial velocity. The z-axis is positively defined upwards while the y-axis is positively defined pointing to the left. In order to speed up the calculation process, only $\frac{1}{8}th$ of the actuator disk is modelled and periodic conditions are applied on the flanks. So instead of obtaining a cylindrical shaped domain, the geometry resembles a "pie-shape" that is extruded downstream as can be seen from Figure 4.1.

Since the calculations performed on these geometries will be used to validate the performance of the UDF, which will be used in a later stage to investigate propeller-wing interaction effects, it must be ensured that the domain extends sufficiently far downstream. In a convential tractor configuration, the wing is placed 1 or more radii behind the propeller. Therefore the limits of the domain are chosen as follows:

- $-2R_p <= x_{origin} <= 4.8R_p$ (axial direction)
- $0 <= r_{origin} <= 3R_p$ (radial direction)

So the domain extends 2 radii upstream, as measured from the origin, 4.8 radii downstream, which is 4.1 radii measured from the outflow plane (corresponds to TE of blade) of the actuator.

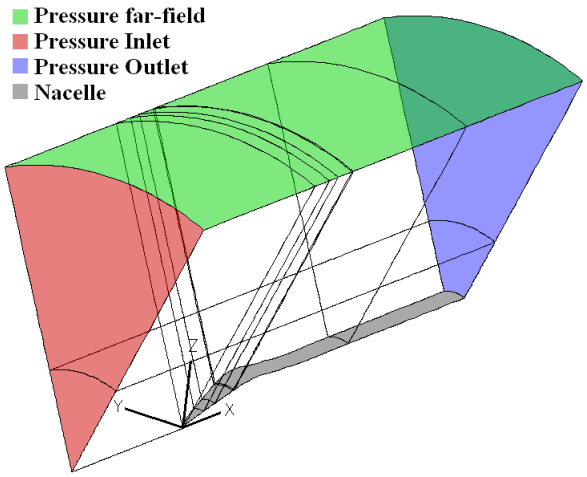


Figure 4.1: 3D view of propeller-nacelle geometry and corresponding nomenclature

In order to validate the results obtained by the actuator disk based simulations, governing flow parameters of the simulations performed on the propeller-nacelle configuration will be compared with Reynolds-Averaged Navier-Stokes (RANS) calculations. These reference data were obtained by B. Marinus, who modelled the propeller blade and nacelle by making use of a sliding mesh (Marinus [2007]). In order to compare the flow variables, plots will be made at different radii positions across the propeller blade. Therefore it is important that the mesh is dense enough to capture all parameters accurately in that part of the domain for which $r < R_p$. Therefore a clear division is made which separates the domain into 2 subdomains: one extending from $r_{nacelle}$ to R_p (lower part) and one extending from R_p to $3R_p$ (upper part). The term $r_{nacelle}$ is the radius measured from the symmetry axis to the nacelle's surface. All "upper part" volumes are subjected to a size function which ensures that the cell size will grow in the radial direction since the mesh is not required to be as dense as in the lower part of the domain. This will reduce the amount of cells needed to fill up the upper part of the domain and thus the calculation time. In axial direction the domain is divided into 7 volumes.

1. volume 1: $-2R_p <= x <= 0$
2. volume 2: $0 <= x <= 0.08\ m$
3. volume 3: $0.08\ m <= x <= 0.13\ m$(LE) (Inflow volume)
4. volume 4: $0.13\ m(LE) <= x <= 0.209\ m$(TE) (Propeller volume)
5. volume 5: $0.209\ m <= x <= 0.215\ m$ (Actuator volume)
6. volume 6: $0.215\ m <= x <= 0.85\ m$
7. volume 7: $0.85\ m <= x <= 3\ m$

Volume 4 and 5 together form the volume beaten by the real propeller. The UDF that will be incorporated in Fluent imposes velocities at the outflow plane of the actuator disk. This is not possible by making use of the density-based solver of FLUENT. Therefore the

pressure-based solver will be used. The difference between the density and pressure-based solver was adressed in chapter 3.2. FLUENT can only impose velocities at boundary faces of a domain, while the actuator outflow plane is an interior face. Therefore one is not able to impose the velocities directly at the outflow plane of the actuator. In order to bypass this problem, velocities will be imposed in the row of cells, which are located just before the outflow plane of the actuator, thus in volume 5. Volume 5 will from now on be referred to as "Actuator volume", while volume 4 will be named "Propeller volume" and volume 3 will be called "Inflow volume", which will be used to read the inflow velocities at the inflow plane of the blade. A side view (xz-plane) of the geometry can be found in Figure 4.2.

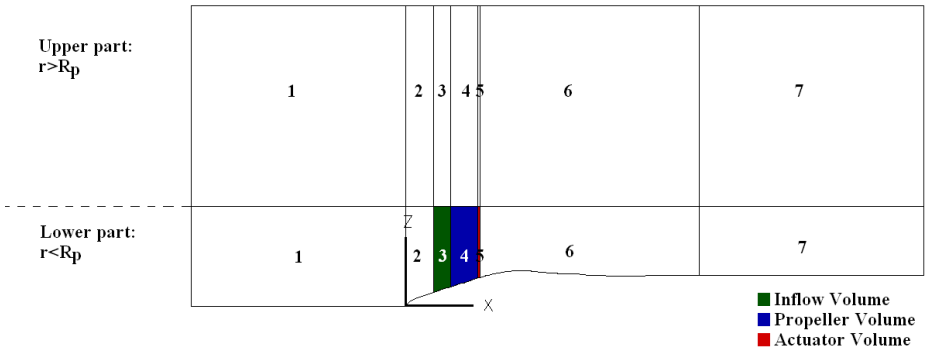


Figure 4.2: Side view of propeller-nacelle geometry and corresponding nomenclature

In order to minimize numerical diffusion hexahedral cells were used as much as possible. If the nacelle is not included the flow is alligned with the mesh until the actuator outflow plane is reached, where the flow is subjected to swirl. However the nacelle is part of the geometry, therefore the flow will already have a radial velocity component when it reaches the nacelle. Especially at the inflow and actuator volume, hexahedral elements are required since the applied UDF needs to extract geometry and flow variables of cells at the same radial distance (Section 4.2). Hexahedral elements ensure that the mesh is more structured, which makes it easier to find centroids located at similar radial positions.

As a first guess a cell height at the inflow plane of the actuator of 0.012 m (in r-direction) was applied, which resulted in 21 cells in radial direction. After a test run of the mesh in FLUENT however it was found that the convergence of the residuals was not satisfactory. Therefore the number of cells in radial direction was doubled to 42, leading to a cell height of 0.006 m at the actuator inflow plane. In order to ensure also a good resolution in axial direction, the length of the actuator volume is 0.006 m which comprises only one column of cells. The total number of cells in tangential direction is 40, which leads to a cell's width of 0.0061 m at $r = R_p$. For the lower part of the domain, the total number of cells in both tangential as radial direction is now fixed as 40 and 42, respectively. The total number of cells in axial direction for the other volumes, except volume 7, of both the lower as upper part of the domain is chosen such that the cell's length within these volumes is close to 0.006 m. A size function in axial direction is imposed at volume 7 in order to reduce the number of cells. All volumes are made up of hexahedron cells, except volumes 1 and 2. Volume 1 is a prism which does not allow a hexahedral meshing type, therefore it is filled with tetrahedral cells. Volume 2 contains tetrahedrons and pyramid

cells. The tetrahedral cells can be found everywhere in volume 2 except at the boundary with volume 3. Here pyramid cells can be found to ensure a smooth transition from tetrahedrons to hexahedron cell types. The mesh of the SR-1 propeller-nacelle geometry can be found in Figure 4.3. A zoom-in of the tetrahedrons in the first lower volume can be found in Figure 4.4

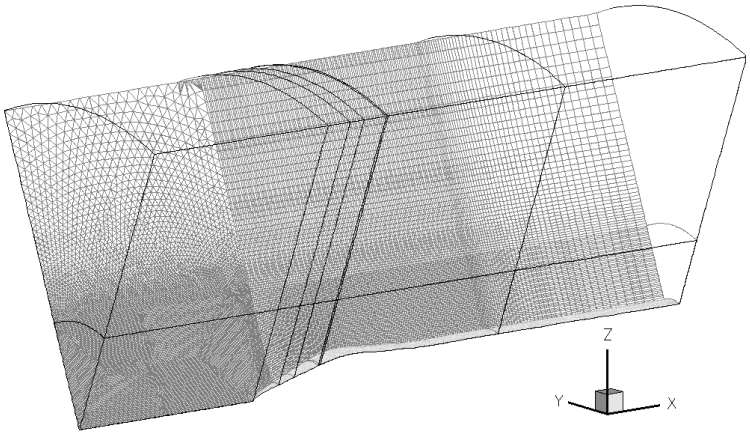


Figure 4.3: Mesh of the SR-1 propeller-nacelle geometry

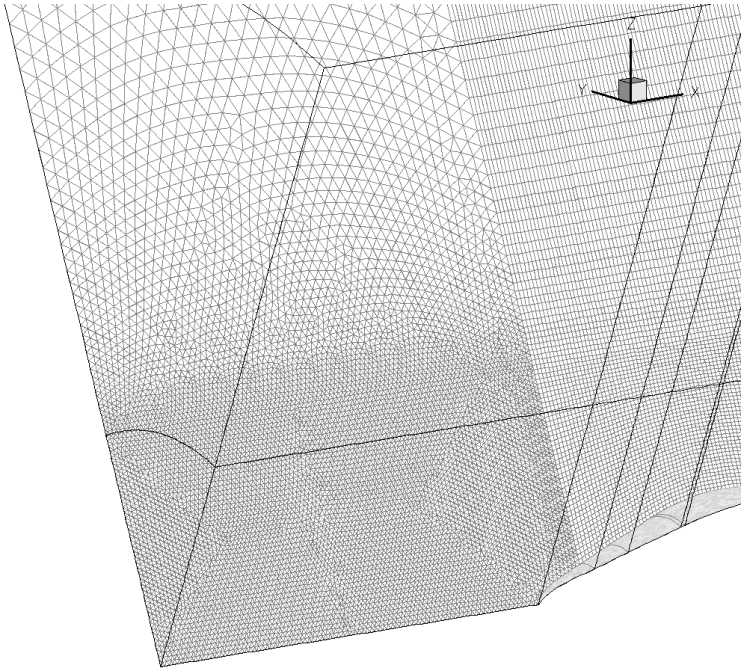


Figure 4.4: Zoom-in of the thetrahedrons of the SR-1 propeller-nacelle mesh

4.2 Set up of the numerical model using a UDF

An actuator disk model imposes a jump in pressure and tangential velocity at the actuator disk. This way of modelling the actuator disk is also available in FLUENT by means of the FAN boundary condition. However, the developed actuator disk model (uniform actuator disk) will impose an axial, tangential and radial velocity at all the cells of the actuator volume to simulate propeller induced flow, by means of Eq (2.108), Eq (2.109) and Eq (2.110). From the discussion in section 2.1.3 it followed that the presence of the nacelle changes the inflow velocity at the propeller blade. This change in inflow velocity will be taken into account by reading out the velocity at the inflow plane of the propeller blade. The source code that takes this non-uniform inflow velocity into account will be referred to as non-uniform actuator disk and is based on the same equations as the uniform actuator disk model except for the fact that the term V_∞ is replaced by V_{in}, the inflow velocity at the blade.
In order to investigate the effect of inclusion of the induced radial velocity, another mathematical model will be included in the validation process. This model neglects the induced radial velocity and is based on Eq (2.89) till Eq (2.104) which are set up for a uniform inflow velocity equal to freestream (Phillips [2004]). This numerical model which does not incorporate the radial velocity induced by the rotating propeller, will be referred to as Phillips' equations.
This chapter will elaborate how these 3 different models are transformed into a User Defined Function (UDF). A UDF is a user-written program that can be dynamically loaded with the FLUENT solver to customize FLUENT to the specific modelling needs of the user. A UDF is written in C-programming language using any text editor and makes use of special macros or commands, which can be accessed by adding the header file udf.h in the source file. These special macros can be split up into 2 categories: the define-macros and the other macros which are used to access cell variables for example. The define macro actually states the main function of the UDF, examples of define macros are:

- DEFINE_ADJUST is a macro that can be used to adjust or modify variables that are not passed as arguments. For example, it can be used to modify flow variables (e.g., velocities, pressure) and compute integrals.
- DEFINE_EXECUTE_AFTER_CASE and DEFINE_EXECUTE_AFTER_DATA are general-purpose macros that can be used to specify a function that executes after the case and/or data file is read in FLUENT.
- DEFINE_PROFILE can be used to define a custom boundary profile that varies as a function of spatial coordinates or time.

Besides accessing cell or face variables, other macros are available which allow for setting boundary condition values and looping over cells, faces or nodes. Since the UDF makes use of macros, it is only applicable for a certain version of FLUENT, since some macros were not yet defined in earlier software versions. The UDF was developed for the 12.0.16 FLUENT version, hence compatibility with older or newer versions of FLUENT can not be ensured. A UDF needs to be compiled and loaded before it can be used. Some define macros, such as DEFINE_PROFILE, need to be hooked to the FLUENT solver

using a graphical user interface dialog box in order to be executed. Since the developed actuator disk model will impose an axial, tangential and radial velocity at all the cells of the actuator volume to simulate propeller induced flow, the written C or source file will consist of 3 different UDF's for all numerical models:

- One that imposes the velocity in x-direction (axial velocity)
- One that imposes the velocity in y-direction (consists of components of the tangential and radial velocity)
- One that imposes the velocity in z-direction (consists of components of the tangential and radial velocity)

The code for the uniform and non-uniform actuator disk can be found in respectively Appendix B.1 and B.2, while the UDF which implements Phillips' equations can be found in Appendix B.3.

All codes start with calling the header files to enable the use of the different macros. After which the constant variables are defined. These constant variables are global variables, meaning that they are known and accessible everywhere in the code. Then the global non-constant variables are defined, such as loop counters and other variables that needs to be known and accessible throughout the entire UDF. Besides global variables, also local variables are available. These local variables are only known and available inside the define macro in which it is declared. For the declaration of a constant, non-constant global or local variable a certain procedure must be followed. A constant global variable is mostly written in capital letters and must be defined as e.g.: "#define RP 0.311", while a non-constant global or local variable does not consist of capital letters and is defined by stating its type, followed by its name and semi-colon e.g.: "int i";. After the declaration of all global variables, the first define macro is used: DEFINE_EXECUTE_AFTER_CASE. The purpose of this define macro is to determine of how many faces the outflow plane of the actuator consists. This variable is called "total_nr_faces_outflowface" and also corresponds to the total number of cells inside the actuator volume since this volume only consists of a depth of 1 cell. The "total_nr_faces_outflowface" parameter is very important since it will be used to declare the arrays of several cell parameters throughout the entire UDF. In order to declare an array, the type and name of the array must be stated together with the number of elements of which the array consists e.g. double r_actuator[total_nr_faces_outflowface]; . By using the DEFINE_EXECUTE_AFTER_CASE macro, the "total_nr_faces_outflowface variable" is determined after reading in the case and hence it is a known variable before the other define macros of the UDF will be executed.
After the DEFINE_EXECUTE_AFTER_CASE macro, the velocity in x-direction will be calculated and imposed at the cells inside the actuator volume. To this end the DEFINE_PROFILE macro is used. From here on the source codes for the 3 different models will differ. First a discussion of the calculation of the velocity in x-, y-, and z-direction will be given concerning the uniform actuator disk model. Since this is the numerical model that will be used in the propeller-wing interaction study, the adjustments of the source code of the other models will be discussed with respect to the uniform actuator disk model and is given in section 4.2.2 and 4.2.3. For these UDF's only that part of the

code differs in which the velocity in x-direction is calculated and imposed. The code for the velocity in y- and z-direction are identical and will therefore not be discussed.

4.2.1 UDF implementation of the calculation of the x-,y- and z-velocities concerning the uniform actuator disk model

UDF imposing the x-velocity

The source code for imposing the x-velocity can be found in Appendix B.1. The DEFINE_PROFILE macro is used to impose the x-velocity and consists of 3 arguments, the UDF name, a general pointer and a general index e.g. DEFINE_PROFILE(x_vel_inboard, t1, indices). Then the local variables will be declared. The global variables which will be calculated in this function and which have already been declared in the global variables declaration part, need to be malloced here. The expression "mallocing a variable" refers to the assignment of memory space for a variable. To ensure that it is only malloced once, meaning that the program will only free memory space for the variable once and not every iteration the following part of code was entered e.g. for the variable V_axial:

```
if(!V_axial)
 {
  Message0("Allocating V_axial\n");
  V_axial = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }
```

After the declaration of the local variables and the mallocing of the global variables, the elemental thrust of each cell inside the actuator volume will be calculated. This is done by implementing Eq (2.92), Eq (2.97) till Eq (2.99) and Eq (2.101) to Eq (2.103) from section 2.3.3. The term dr_end has a constant value, when the validation simulations are considered since it denotes the height of a cell in radial direction inside the actuator volume. However for the propeller-wing interaction study a boundary layer will be applied which will lead to a variable cell height inside the actuator volume (See Figure 4.5).

To take this variation of the term dr_end into account, a general calculation method for this term was developed, making it applicable to both the validation simulations as the propeller-wing interaction study. After all, the validation simulations are performed to ensure that the numerical model is accurate enough, thus ensuring that it can be used to study the propeller-wing interference effects. The determination of the term dr_end will be elaborated below.
Consider a cell inside the actuator volume as represented in Figure 4.6. The macro C_VOLUME can be used to determine the volume of each cell inside the actuator volume. The length of each cell in x-direction, dx is known and is equal to the thickness of the actuator volume. The width of a cell in tangential direction, denoted as dy can be calculated by dividing the circumference of the actuator disk at that radius position by the total number of cells in tangential direction, nr_arc_cells. The height, dr_end can then simply be determined by dividing the cell's volume by the cell's area ($= dx \cdot dy$). Although the edges of the cell in tangential and axial direction are curved, it is assumed that the upper and lower side for both the axial and tangential direction have the same amount

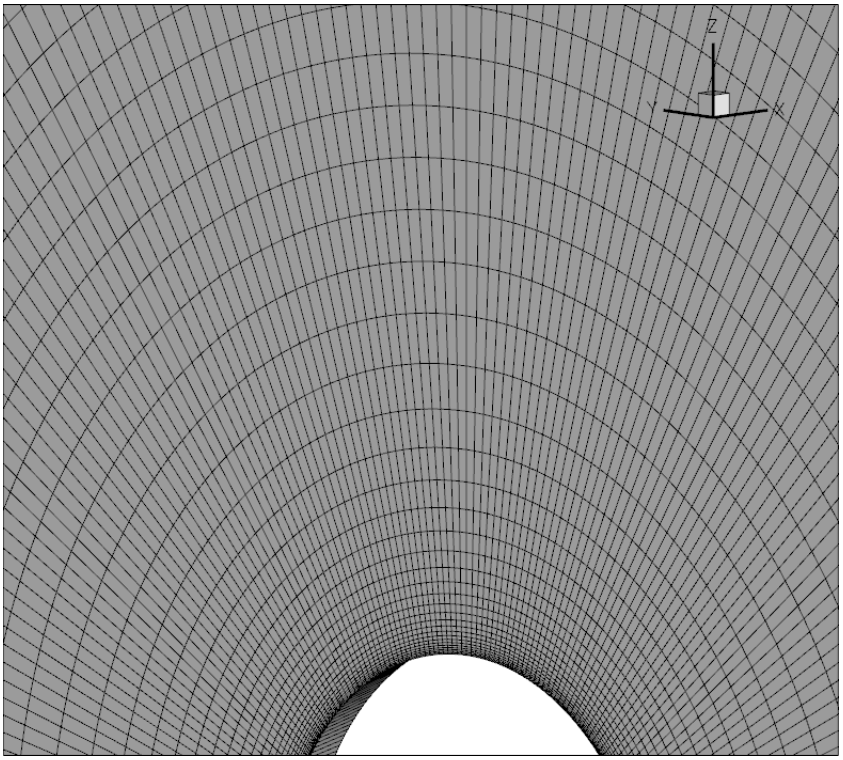


Figure 4.5: Variation of the height of a cell in radial direction

of curvature, allowing the above stated calculation procedure. The parameter dr_end is crucial for the calculation of the elemental thrust and hence for the determination of the induced axial velocity. In order to apply this method for the determination of dr_end the cells of the mesh inside the actuator volume have to be of hexahedral type. This illustrates the close coupling between mesh and UDF.
To check whether the calculation of the elemental thrust is correct, a short verification loop is set up, which sums the elemental thrusts over all the cells of the actuator volume. Since the cells on the same radius have the same elemental thrust and Eq (2.92) is valid for an annular ring. The total elemental thrust is found by dividing the sum of the elemental thrust of all cells by the term divider. This term is the division of the total_nr_faces_outflowface/total_nr_cells_z, in which the term total_nr_cells_z denotes the total amount of cells in radial direction inside the actuator volume. Now that the elemental thrust of each cell is known, the axial velocity can be determined. This is done by using Eq (2.104), Eq (2.106) and Eq (2.107) from section 2.3.3. Since at this point in the UDF all induced axial velocities are known, the sum of all these induced axial velocities (sum_v_A) is taken since it will be needed in the calculation of the induced radial velocity.

UDF imposing the y-velocity

For imposing the y-velocity, the DEFINE_PROFILE macro is used. First the global variables are malloced and the local variables are declared. Then a loop is initiated in which the radius of the centroid of each cell is calculated, together with its corresponding y- and z-coordinate. Since the y-velocity consists of the y-component of the tangential

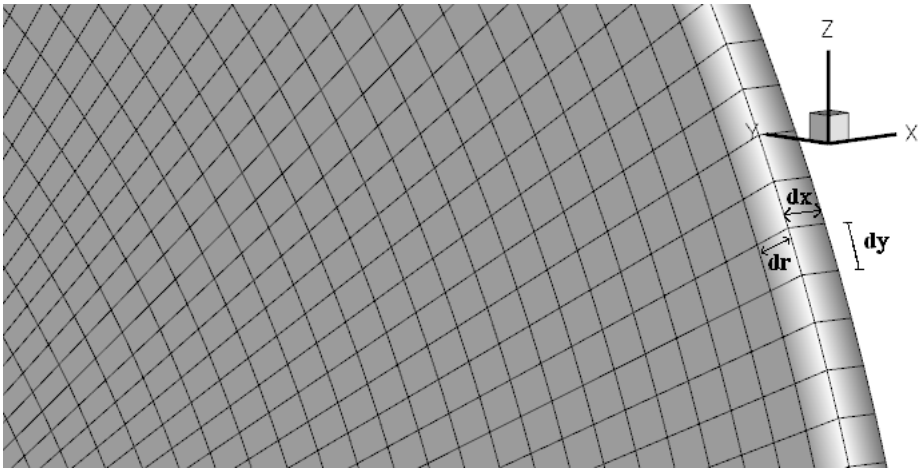
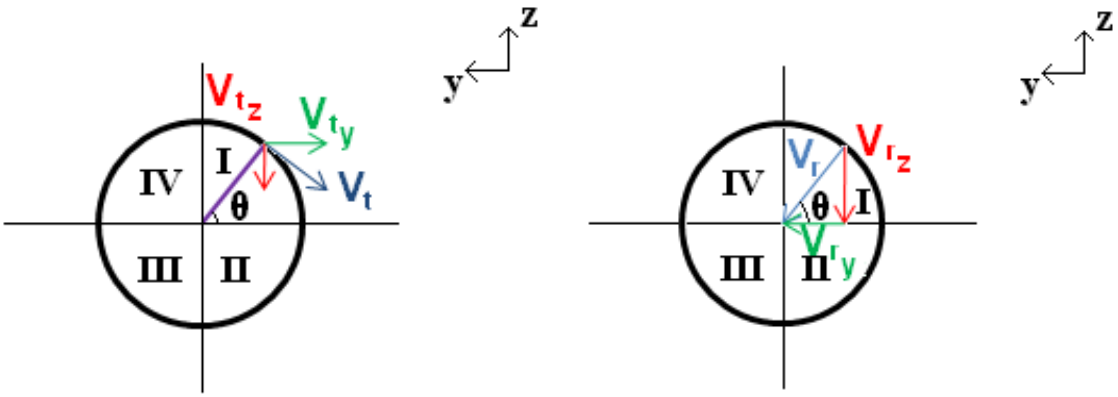


Figure 4.6: Calculation of the height of a cell in radial direction

and radial velocity, the tangential and radial velocities must be calculated first. This is done by making use of Eq (2.109) and (2.110), respectively. The decomposition of the tangential and radial components into their y- and z-component for a clockwise rotating propeller (when viewed from the front) can be found in Figure 4.7a and 4.7b respectively.



(a) Tangential velocity

(b) Radial velocity

Figure 4.7: Decomposition of the tangential and radial velocity into its x- and y-component for a clockwise rotating propeller

Suppose that Figure 4.7 gives a schematic representation of an clockwise rotating propeller, which has its hub point positioned at coordinates (x_E, y_E, z_E) which does not coincide with the origin of the system. The actuator disk can then be split up in 4 different quadrants indicated by the Roman numbers I, II, III and IV. The sign of the angle θ differs for each region. To generalize the expressions for V_{t_y} and V_{t_r} the absolute value of θ will be taken:

$$\theta = \begin{cases} \arctan\left|\frac{z-z_E}{y-y_E}\right| & \text{If (y} \neq y_E) \\ \frac{\pi}{2} & \text{else} \end{cases}$$

Both the tangential and radial component will now be decomposed in their y- and z-velocity components. Since the variables V_{t_z} and V_{r_z} are global variables, they can easily be called in the UDF that imposes the z-velocity, hence eliminating the need to calculate them again. The components of the tangential and radial velocities are then obtained from the equations in Table 4.2.

Table 4.2: Equations expressing the y- and z-velocity for a clockwise rotating propeller

Region I	**Region II**
$V_{t_y} = -V_t \cdot \sin\theta$ $V_{t_z} = -V_t \cdot \cos\theta$ $V_{r_y} = V_r \cdot \cos\theta$ $V_{r_z} = -V_r \cdot \sin\theta$	$V_{t_y} = V_t \cdot \sin\theta$ $V_{t_z} = -V_t \cdot \cos\theta$ $V_{r_y} = V_r \cdot \cos\theta$ $V_{r_z} = V_r \cdot \sin\theta$
Region III	**Region IV**
$V_{t_y} = V_t \cdot \sin\theta$ $V_{t_z} = V_t \cdot \cos\theta$ $V_{r_y} = -V_r \cdot \cos\theta$ $V_{r_z} = V_r \cdot \sin\theta$	$V_{t_y} = -V_t \cdot \sin\theta$ $V_{t_z} = V_t \cdot \cos\theta$ $V_{r_y} = -V_r \cdot \cos\theta$ $V_{r_z} = -V_r \cdot \sin\theta$

For a counter-clockwise rotating propeller only the sign of the terms V_{t_y} and V_{t_z} change. Define the parameter "sense", which is 1 for a clockwise rotating propeller and 0 for a counter-clockwise rotating propeller. The expressions for a clockwise and counter-clockwise rotating propeller can now be grouped as follows:

Table 4.3: General equations expressing the y- and z-velocity for clockwise and counter-clockwise propellers

If $z < z_E$	If $z > z_E$
$V_{t_y} = (-1)^{sense+1} V_t \cdot \sin\theta$ $V_{r_z} = V_r \cdot \sin\theta$	$V_{t_y} = (-1)^{sense} V_t \cdot \sin\theta$ $V_{r_z} = -V_r \cdot \sin\theta$
If $y < y_E$	If $y > y_E$
$V_{t_z} = (-1)^{sense} V_t \cdot \cos\theta$ $V_{r_y} = V_r \cdot \cos\theta$	$V_{t_z} = (-1)^{sense+1} V_t \cdot \cos\theta$ $V_{r_y} = -V_r \cdot \cos\theta$

After the determination of the y-components of the tangential and radial velocity, the velocity in y-direction can be evaluated with Eq (4.1).

$$V_y = V_{t_y} + V_{r_y} \tag{4.1}$$

UDF imposing the z-velocity

As was the case with the x- and y-velocity, the DEFINE_PROFILE macro is used to calculate and impose the z-velocity. After mallocing the global variable V_z, the velocity in z-direction is determined by means of formula (4.2):

$$V_z = V_{t_z} + V_{r_z} \tag{4.2}$$

4.2.2 UDF implementation of the calculation of the x-velocity concerning the non-uniform actuator disk model

The UDF that will be discussed in this subsection is valid for a non-uniform inflow velocity at the propeller blade and is based on an older version of the uniform actuator disk UDF. In this older version of the UDF it was assumed that the variable dr_end was constant, which is only true for the validation mesh. Hence the term dr_end can be calculated as:

$$dr\_end = \frac{R_p - R_{TE}}{total\_nr\_cells\_z} \tag{4.3}$$

with: R_{TE} = radius of the nacelle at the trailing edge of the propeller blade

The source file for the non-uniform actuator disk can be found in B.2. After the declaration of the local and global variables, the variables C_1 and C_2 are imported. These variables are the constants of the linearized relation that describes the pathline of a particle when traveling over the propeller hub. These variables will be used to find out which inflow velocity belongs to which cell at the outflow plane of the actuator volume. It is assumed that this relation is axisymmetric, since the nacelle is also axisymmetric. These variables are obtained by drawing the plane y=0 in FLUENT and intersecting this plane with the inflow and outflow plane of the actuator volume, creating 2 vertical lines. From the line at the inflow plane of the actuator volume, particle lines are released. The information regarding the particle lines are stored in the file *pathline.fvp*, furthermore the x, y, and z-coordinates of the centroids of the cells lying on the line at the inflow and outflow plane are extracted as well. By means of the particles path lines a relationship is found between the inflow radius at the centroid of the cells and the outflow radius, which does not necessarily correspond to the centroid of a cell, see Figure 4.8.

It is then assumed that the relationship between the outflow radii that do correspond to a cell's centroid and the inflow radii which are not necessarily located at the centroids of the cells at the inflow plane is the same. So the terms C_1 and C_2 can then be used to calculate the radius of the cells at the inflow plane which corresponds to the radius of the corresponding cell at the outflow plane. Since it is impossible to read out face variables if the face does not represent a boundary, the inflow velocity is read at the last column of cells of the inflow volume. Hence the velocity at the inflow plane is approximated by reading out the velocity at the cells just before this plane. Since the nacelle geometry is axisymmetric around the x-axis, it is possible to collect the cells that lie on the same radius. Therefore the radii of the cells are compared to each other to find total_nr_cells_z different radii with their corresponding inflow velocity. These arrays are then sorted in an ascending order since the C_1 and C_2 parameters are also sorted in an ascending order and consists of total_nr_cells_z different entries. The radii of the cells inside the actuator

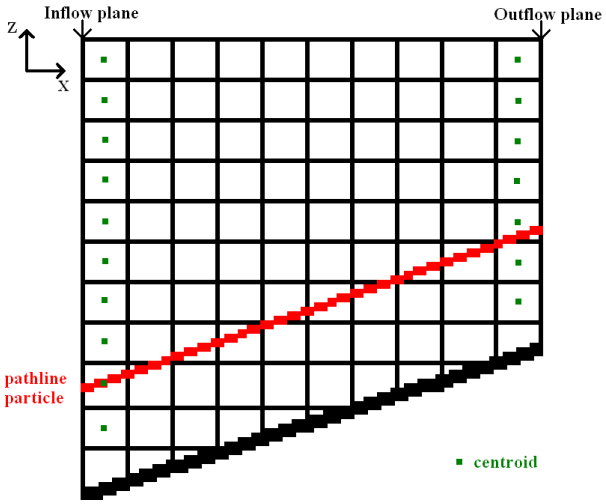


Figure 4.8: Relation between the inflow velocity and the outflow plane

volume are also extracted and compared to each other in order to have total_nr_cells_z different outflow radii and are also sorted in ascending order. Then the term r_begin is calculated using the C_1 and C_2 terms. The corresponding inflow velocity V_{in} is then found by interpolating the values that have been read at the inflow plane. The elemental thrust can then be found by using Eq (2.92), Eq (2.97) till Eq (2.99) and Eq (2.101) to Eq (2.103) from section 2.3.3. The axial velocity can now be calculated by using Eq (2.104), Eq (2.106) and Eq (2.107). As was the case with the uniform UDF, the term sum_v_A is also determined in this part of the code. Finally a loop is created to couple the correct inflow, axial and axial induced velocity to the corresponding radius since until now these parameters were only known for total_nr_cells_z different radii, but they need to be know for all the cells of the actuator volume. Finally, the axial velocity is imposed at all the cells of the actuator volume.

4.2.3 UDF implementation of the calculation of the x-velocity concerning Phillips' equations

The UDF based on Phillips' equations is set up in the same way as the non-uniform actuator disk UDF, except for the equations concerning the axial, tangential and radial velocity. Furthermore the non-uniform inflow velocity is no longer taken into account, so all terms related to the inflow velocity are replaced by V_∞. The induced radial velocity is neglected, hence the y- and z-velocity only consist of the components of the tangential velocity which is calculated by means of Eq (4.4).

$$V_t = \omega_s r_s \tag{4.4}$$

The axial velocity is calculated by means of Eq (4.5).

$$V_a = V_\infty + v_i \tag{4.5}$$

where the expression of v_i given by Eq (2.99).

For more information regarding the set up of the UDF, which incorporates Phillips' equations, the reader is referred to Appendix B.3.

4.3 FLUENT settings involving the validation simulations

This section will discuss the settings that were applied in FLUENT for the validation simulations.

4.3.1 Model settings

A 3-dimensional, steady, compressible, turbulent flow without heat transfer will be modelled. The "Realizable $k-\varepsilon$" method is chosen as turbulent model with "Non-equilibrium wall functions". See section 3.2.1 for more information concerning this turbulence model.

4.3.2 Material properties

The material properties of the fluid (air) are default except for the viscosity, which is set to $1.7819 \cdot 10^{-}5 \frac{kg}{ms}$. This value is obtained by Sutherland's law in order to simulate a correct $Re_{75\%c}$ correponding to a freestream Mach number of 0.6 at an ISA altitude of 35000 ft. Furthermore it is treated as an ideal gas. The nacelle material is set to aluminum and the default values are used.

4.3.3 Solver controls

As stated previously, the pressure-based solver is used since the density-based solver does not allow the attachment of a UDF in order to impose the velocities in the actuator volume. Both solvers employ finite-volume discretization method, but the approach to linearize and solve the discretized equations is different. The velocity field of the solution is by both methods obtained from the momentum equations. However the pressure field is determined differently. The density-based solver determines the pressure field by means of the Equation of state. On the other hand the pressure-based solver calculates the pressure field by solving a pressure or pressure correction equation, which is obtained by manipulating the continuity and momentum equations in such a way that the velocity field, which is corrected by the pressure equation, satisfies the continuity equation. The pressure-based solver algorithm is chosen to be segregated. This implies that the individual governing equations for the solution variables (e.g. V, p, T) are solved one after another. In contrast, the coupled algorithm solves the governing equations for the solution variables simultaneously. Therefore it is not suitable for UDF employment, which imposes velocities. Since it calculates the values for velocity and pressure at the same moment and hence it ignores the UDF since it already has imposed a velocity together with the pressure at a cell. The SIMPLE algorithm is chosen for the pressure-velocity coupling.
There are different discretization schemes available for the pressure, density, momentum,

energy, turbulent kinetic energy and turbulent dissipation rate. For the validation simulations the first order discretization method is assumed to be accurate enough. The gradient terms are determined by means of the Green-Gauss cell-based calculation method.

4.3.4 Boundary conditions

For the flanks of the domain, rotationally periodic boundary conditions are imposed in order to obtain the same results as when one would model a complete actuator. The top faces of the domain are categorized as pressure-far-field boundary conditions, which is used to model freestream conditions at infinity. The freestream Mach number (0.6) and direction must be given as input as well as the freestream static temperature (218.81 K). The inlet is modelled as a pressure-inlet on which a gauge total pressure of 6800 Pa is employed (Eq 3.33) and a total temperature of 234.56 K (Eq (3.34)), while the outlet is modelled as a pressure outlet, with zero gauge static pressure and radial pressure distribution. The difference in gauge pressure between the inlet and outlet will produce a flow with Mach number 0.6. The nacelle surface is modelled as an adiabatic no slip wall, while all the other faces are stated to be interior faces. Since the flow is turbulent, turbulence parameters need to be specified for all boundaries of the domain. The turbulence parameters are the turbulent intensity and length scale, which are chosen as 1% and 0.05 m respectively.

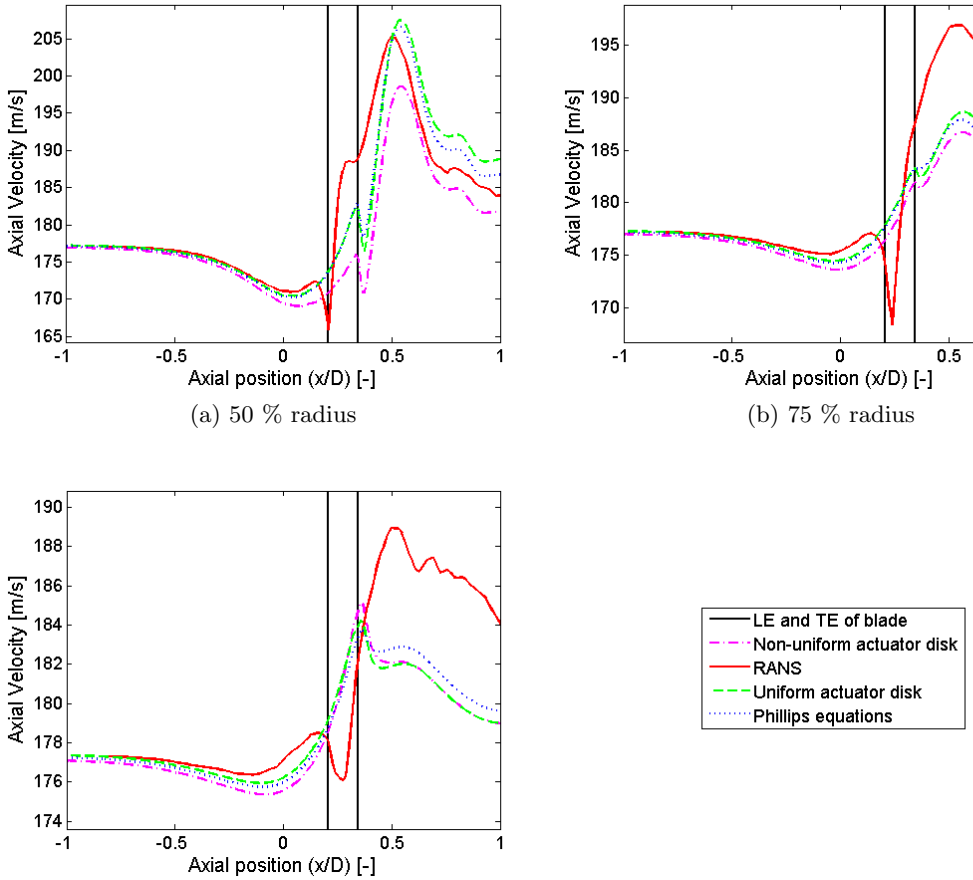
4.4 Discussion of the results of the validation simulation

This section will discuss the results for the simulation performed with the uniform and non-uniform actuator disk model, developed in section 4.2 for the SR-1 propeller-nacelle configuration. In order to validate the results of these numerical models, a simulation was performed for M=0.6 & J=3.08 at an altitude of 35 000 ft or 10 668 m and results for different flow parameters (M, ρ, etc.) are shown at 3 different radius locations along the blade (50%R_p, 75%R_p and 96%R_p). The only data, needed to perform the simulation, is the total thrust of the propeller, the rotational speed, geometric and mesh data such as the radius and number of cells in radial direction. Therefore the model is suitable during the preliminary design of an aircraft when propeller data is scarce and subject to change. As stated before, 3 different numerical models are used in the validation simulation: the uniform and non-uniform actuator disk model and Phillips' equations. The results of these simulations will be compared to each other, but also with the full-blade modelling results using RANS.
The wing of an aircraft is placed one or more radii behind the propeller for most tractor configurations. Since the goal of this thesis is to study propeller-wing interaction effects, it is sufficient that the results obtained by the uniform and/or non-uniform actuator disk simulations agree well with the Reynolds-averaged Navier-Stokes simulations for $x > 0.53\ m$. In case one would be interested in studying propeller-overlap configurations, the afore mentioned models need to agree well with the RANS simulations for $x => 0.215\ m$, which corresponds to the outflow plane of the actuator (TE of the blade). In order to assess the performance of the non-uniform and uniform actuator disk model, the Mach number, axial, radial and tangential velocity, total and static pressure and density plots

at 50%, 75% and 96% radius for the different numerical models will be given below. Sometimes errors and reference values are given to illustrate the difference or similarity in performance of the different models. These errors or reference values are considered at 1 radius behind the trailing edge of the propeller blade. The black vertical lines which are present in all figures represent the LE and TE of the actuator.

4.4.1 Variation of Axial velocity with axial position

As can be seen from Figure 4.9, Phillips equations show the best agreement with the RANS simulation for 50 and 96% radius. For 50% radius, the uniform actuator disk and Phillips' equations display the same behaviour, they only differ in the error w.r.t. the RANS results: for one radius behind the propeller their error is respectively 1% and 2%. The error is the largest at 75 % radius for all numerical models, however the difference between the non-uniform and Phillips equations is small. Downstream of the TE, the error with respect to the RANS simulations decreases with increasing axial position for all models. Far upstream of the LE all flows show a similar behaviour, however just upstream of the LE all flows deviate from the RANS simulation. This is probably caused by the fact that the RANS simulation models the blade surfaces as a wall, while they are modelled as interior faces for all the other simulations. This is also the cause of the bad modelling of the axial velocity inside the actuator disk. So the modelling inside the actuator disk should be ignored for all figures.



(a) 50 % radius

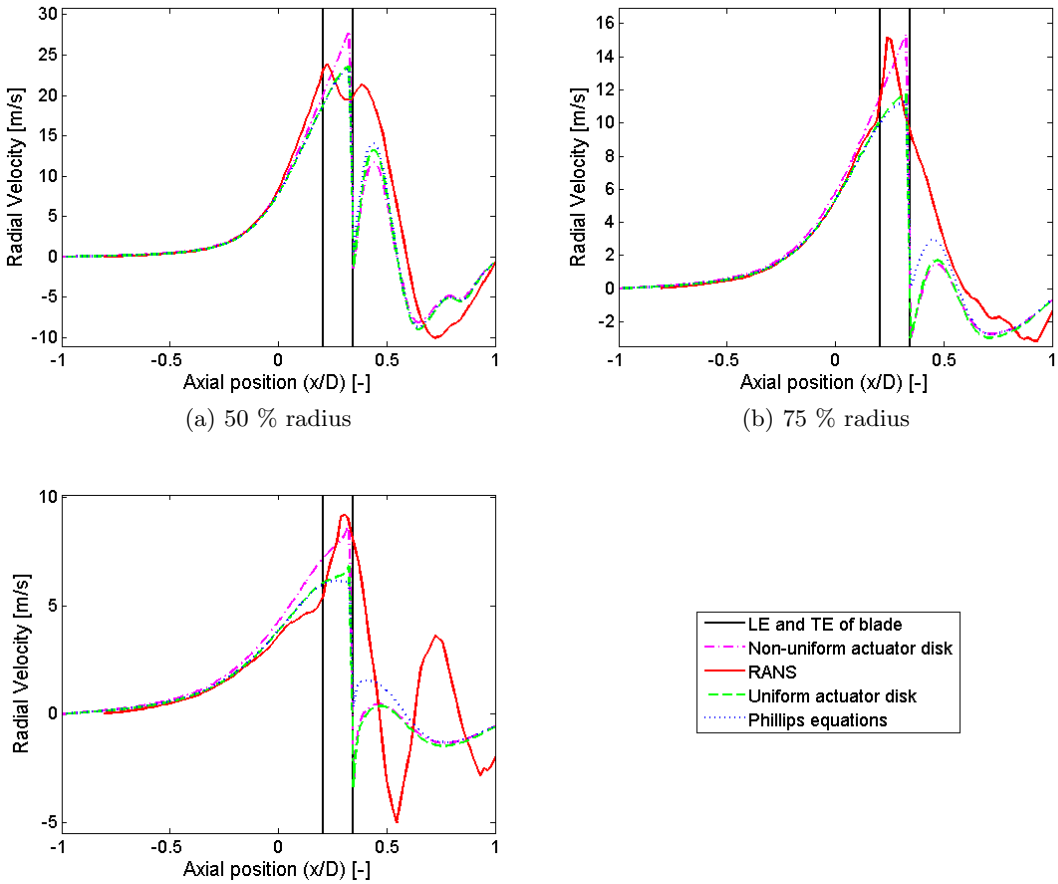
(b) 75 % radius

(c) 96 % radius

Figure 4.9: Axial velocity variation with axial position for M=0.6 & J=3.08.

4.4.2 Variation of Radial velocity with axial position

As can be seen from Figure 4.10, the non-uniform and uniform actuator disk model produce similar results. At one radius behind the TE and further downstream, all models produce identical results for all radii. This is quite surprising since the model based on Phillips' equations does not incorporate the induced radial velocity in the derivation of its formulae. As stated before, the flow upstream of the LE of the actuator disk is modelled quite accurately, until just in front of the LE. The under- or overestimation of the radial velocity of the actuator disk simulations downstream of the actuator disk is caused by the fact that they do not include the effect of the tip vortices while the RANS similation does.



(a) 50 % radius

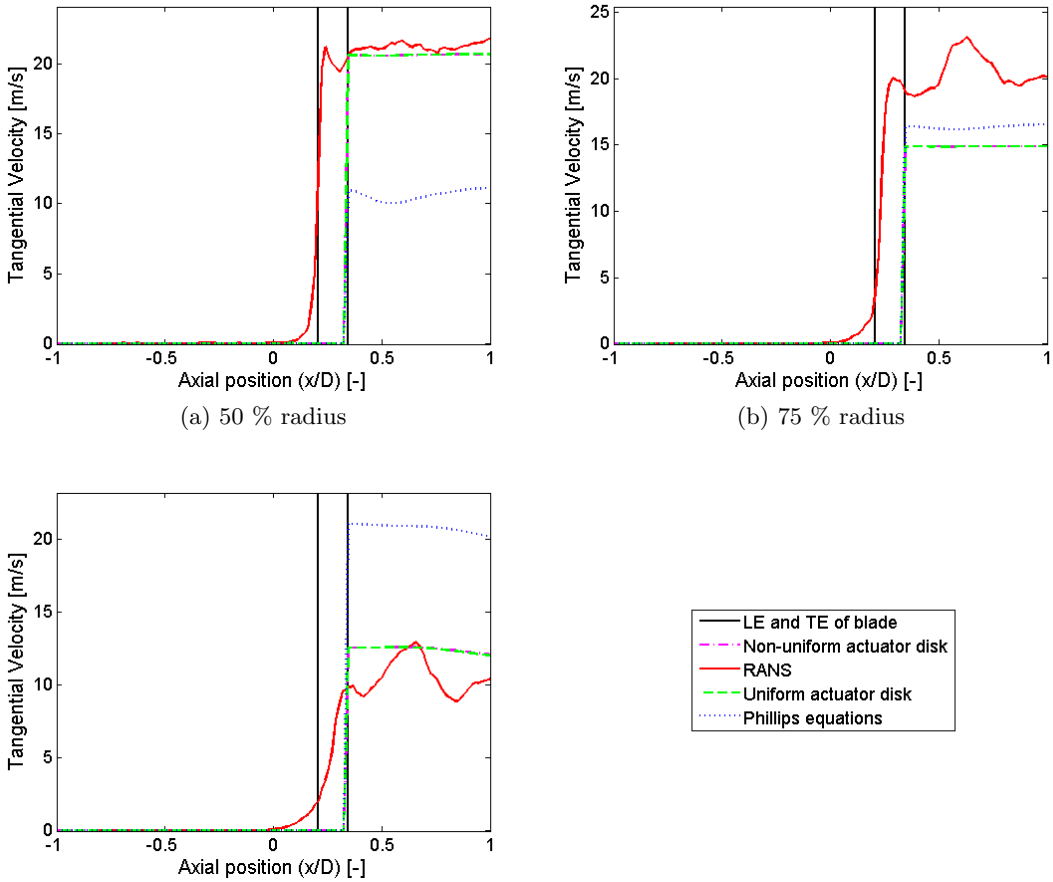
(b) 75 % radius

(c) 96 % radius

Figure 4.10: Radial velocity variation with axial position for M=0.6 & J=3.08.

4.4.3 Variation of Tangential velocity with axial position

As can be seen from Figure 4.11, the non-uniform and uniform actuator disk model produce identical results. Both actuator disk models produce more accurate results than Phillips' equations. Except at 75% radius where Phillips' equations show a slightly better agreement with the RANS results, however the difference between Phillips' and both actuator disk models is only 1.5 m/s. Since the uniform and non-uniform actuator disk models produce an error within 0.5 m/s for 50% radius and for 75 and 96% radius an error smaller than 3.5 m/s when compared to the RANS simulations they definitely outperform Phillips' equations which produce a difference of 10 m/s for 50 and 96% radius and an error of 3 m/s at 75% radius.



(a) 50 % radius

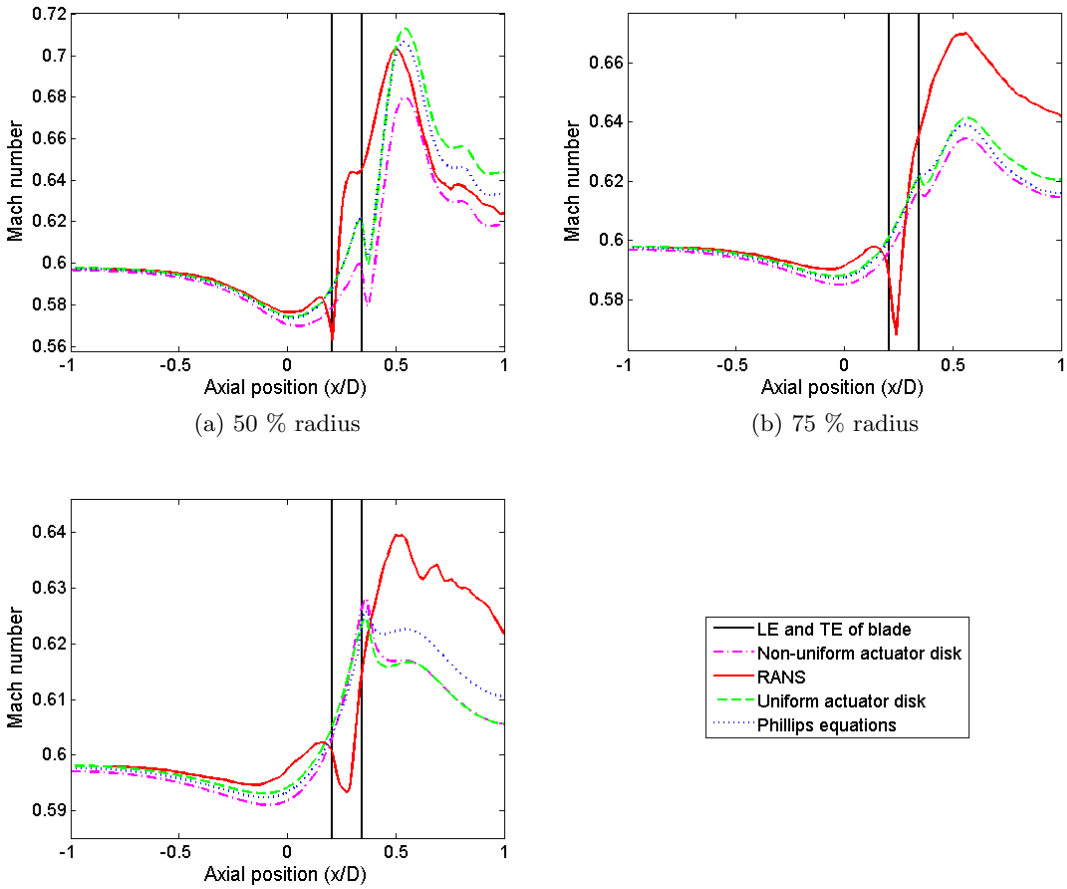
(b) 75 % radius

(c) 96 % radius

Figure 4.11: Tangential velocity variation with axial position for M=0.6 & J=3.08.

4.4.4 Variation of Mach number with axial position

By inspection of Figure 4.9 and 4.12 it can be seen that the variation of the Mach number is similar to the variation of the axial velocity. This makes sense since the radial and tangential velocity are small compared to the axial velocity. Hence it is primarily only the axial velocity that influences the Mach number. Therefore the same conclusions apply as were drawn in section 4.4.1.



(a) 50 % radius

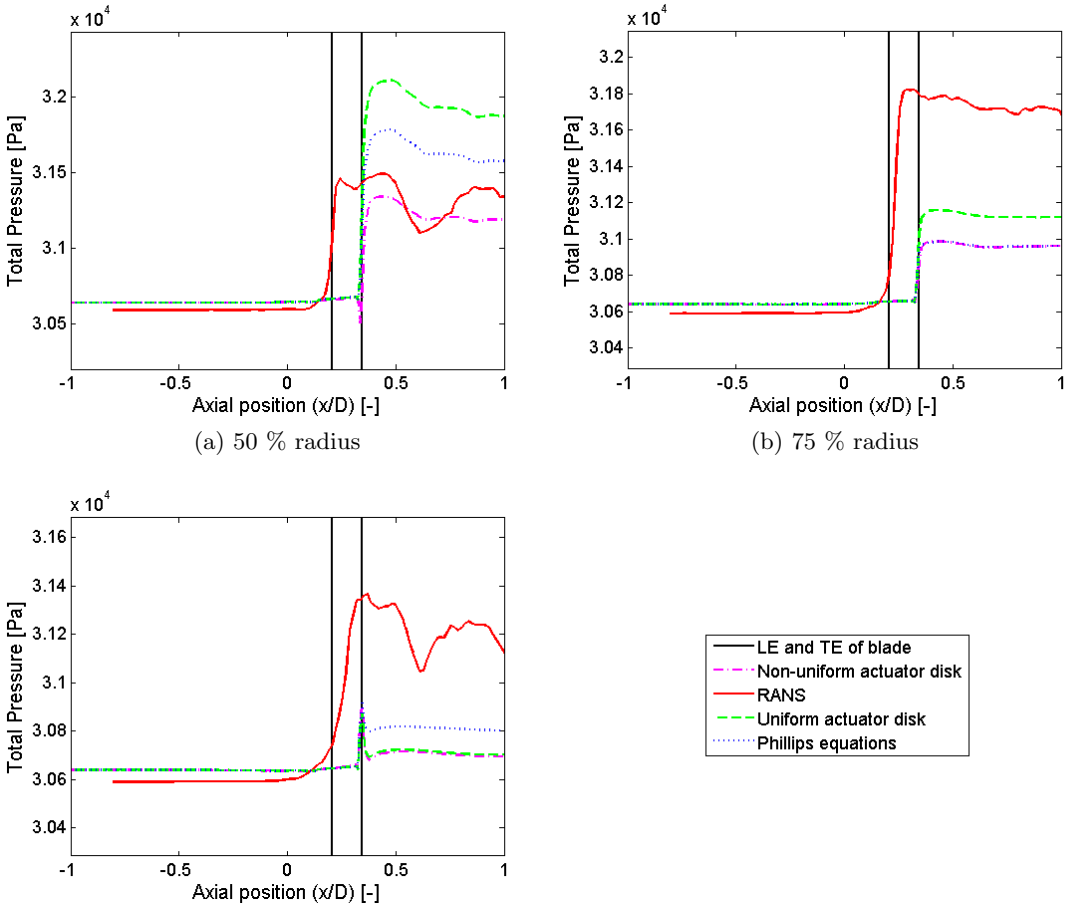
(b) 75 % radius

(c) 96 % radius

Figure 4.12: Mach number variation with axial position for M=0.6 & J=3.08.

4.4.5 Variation of total pressure with axial position

From Figure 4.13 it can be seen that Phillips' equations correspond the best with the results obtained by RANS for 96% radius. However the difference in error is small for all radii. For 96% radius the error of all simulations are below 2%, while at 75% radius the error stays lower than 2.5%. For 50% radius, the error is 0.7% for Phillips' equations and 0.6 and 1.7% for respectively the non-uniform and uniform actuator disk models.



(a) 50 % radius

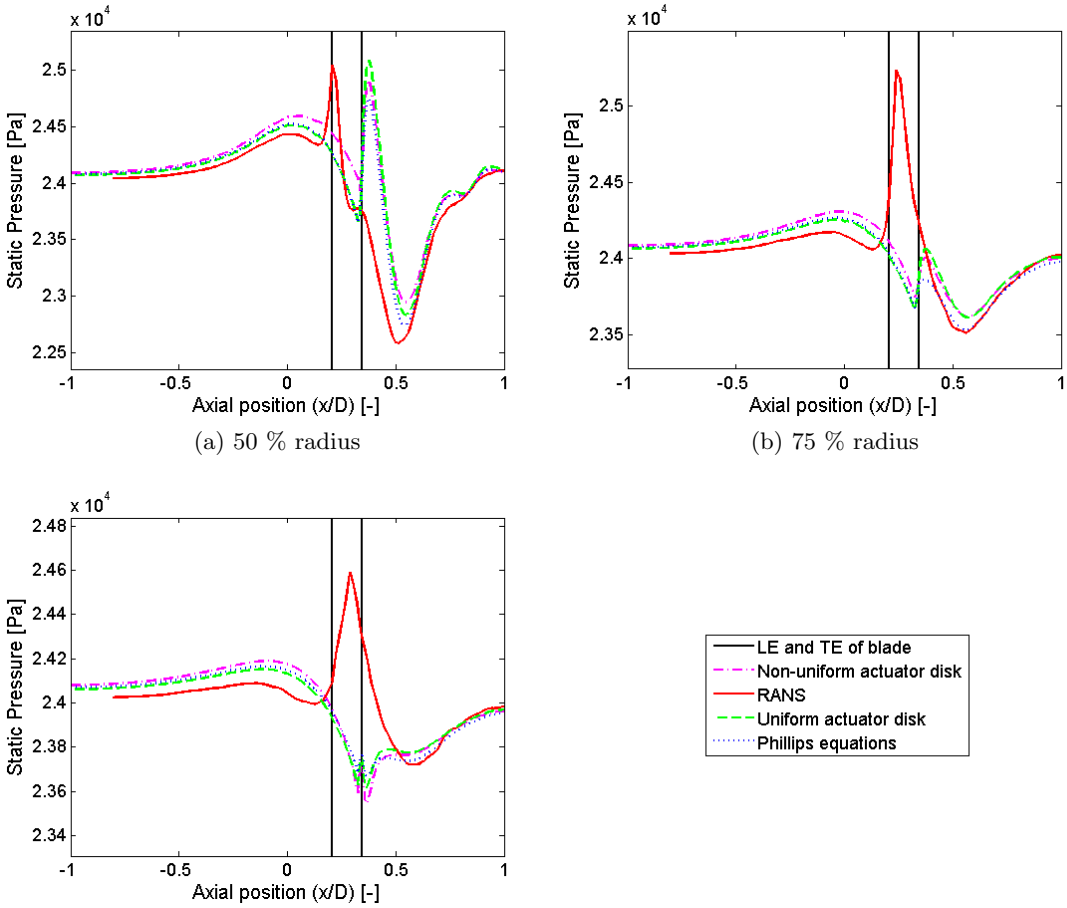
(b) 75 % radius

(c) 96 % radius

Figure 4.13: Total pressure variation with axial position for M=0.6 & J=3.08.

4.4.6 Variation of static pressure with axial position

All numerical models show a small difference in static pressure when compared to the RANS simulations as can be seen from Figure 4.14. However the error introduced is so small that it can be ignored.



(a) 50 % radius

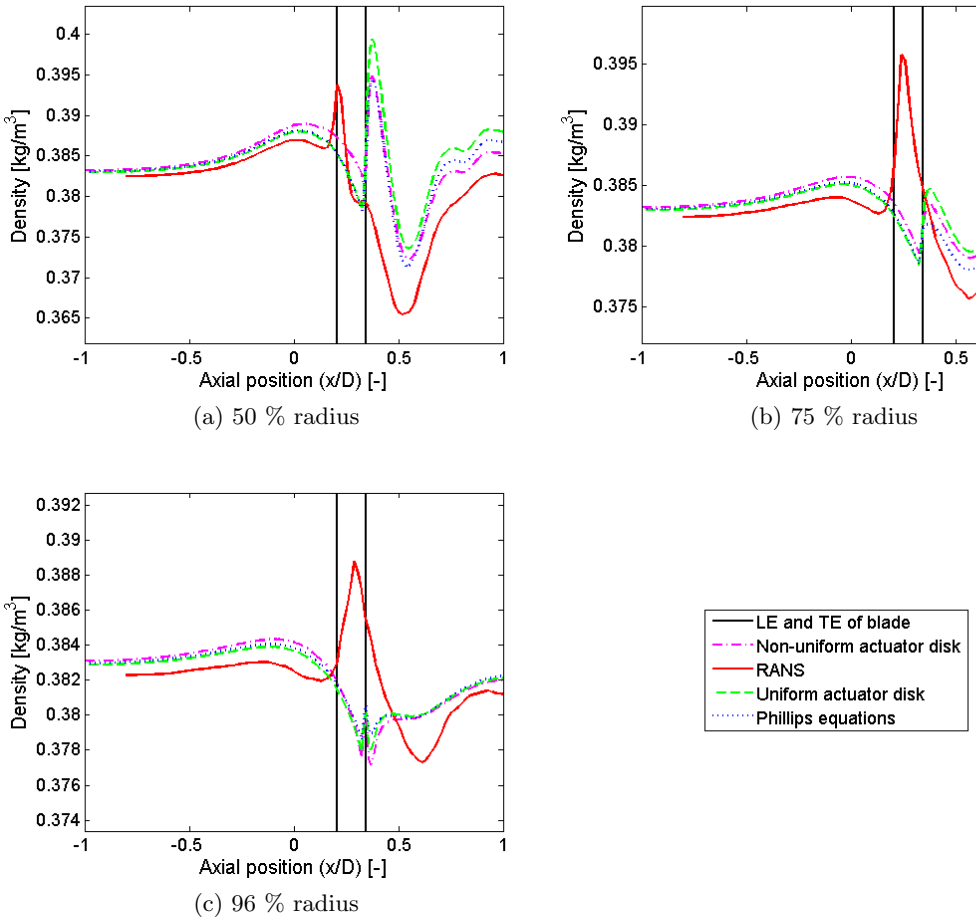
(b) 75 % radius

(c) 96 % radius

Figure 4.14: Static pressure variation with axial position for M=0.6 & J=3.08.

4.4.7 Variation of density with axial position

From Figure 4.13 it can be seen that there is a small discrepancy in density between all numerical models when compared to the RANS simulations. However the error introduced is so small that it is negligible.



(a) 50 % radius

(b) 75 % radius

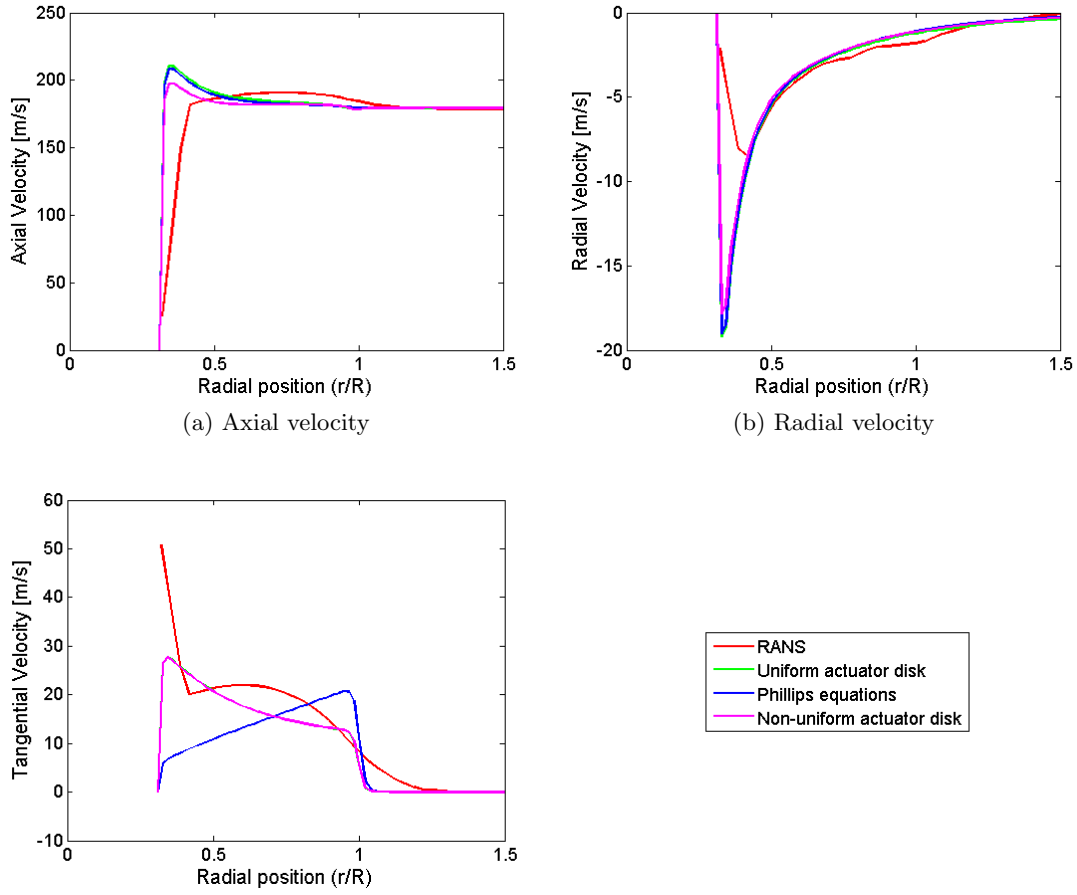
(c) 96 % radius

Figure 4.15: Density variation with axial position for M=0.6 & J=3.08.

4.4.8 Comparison of axial, radial and tangential velocity distribution in radial direction

Figure 4.16 represents the axial, radial and tangential velocity distribution in radial direction of the different numerical models at 1 radius behind the trailing edge of the blade, so at $x = 0.526$ m or $\frac{x}{R} = 1.69$. To this end the data concerning the RANS simulation is averaged circumferentially. This is done by extracting data along 11 different radial lines in the plane $\frac{x}{R} = 1.69$ and then averaging it by taking the mean value.
It can be seen from Figure 4.16a that both actuator disk models as Phillips' equations agree well with the axial velocity distribution predicted by the RANS simulations. For $0.3 <= x/r <= 0.5$ the non-uniform actuator disk model provides the best agreement with the RANS results, while for $0.5 <= x/r <= 0.8$ the uniform actuator disk model shows the best agreement.
Concerning the radial velocity, all simulations agree well with the RANS simulation. However it appears that the uniform actuator model shows a slightly better agreement, although the difference is small and decreases with increasing radial position.
The strong point of the uniform and nonuniform actuator disk model is apparent in Fig-

ure 4.16c, where it can be clearly seen that the tangential velocity is better predicted than Phillips' equations. However the tangential velocity on the nacelle's surface is underestimated for both actuator disk models and Phillips' equations. This proves that the actuator disk model can not take the viscous entrainment effect into account. However further away from the spinner's surface, the prediction agrees quite well with the RANS results.
The better agreement of the tangential velocity and a good representation of the axial and radial velocity makes the uniform and non-uniform actuator disk model superior with respect to Phillips equations.



(a) Axial velocity

(b) Radial velocity

(c) Tangential velocity

Figure 4.16: Distribution in radial direction for at $\frac{x}{R} = 1.69$ for M=0.6 & J=3.08.

4.5 Choice of numerical model

From the previous section, the following general conclusions can be drawn:

- The uniform actuator disk model gives more accurate results than the non-uniform actuator disk model for $x < x_{LE}$. For $x > x_{TE}$ these models produce similar results.

- The uniform and non-uniform actuator disk model give a better prediction of the tangential velocity at 50% and 96% radius than Phillips' equations. For 75% radius the results are similar.
- The non-uniform actuator disk model shows the best agreement with the axial velocity of the RANS simulations at one radius behind the TE of the propeller for $0.3 <= x/r <= 0.5$, while for $0.5 <= x/r <= 0.8$ the uniform actuator disk model provides the best prediction.
- The radial velocity distribution at one radius behind the propeller is slightly better predicted by the uniform actuator disk model when compared to the other models.
- The tangential velocity distribution at one radius behind the propeller is much better predicted by the uniform and nonuniform actuator disk models than Phillips' equations.

The uniform and nonuniform actuator disk models produce similar results. Surprisingly the prediction of the radial velocity obtained by means of Phillips' equations is only slightly worse than the prediction made by both actuator disk models although the radial induced velocity is neglected in the derivation of Phillips' equations. However the uniform and nonuniform actuator disk model outperform Phillips' equations when considering the prediction of the tangential velocity.

Similar simulations were performed with Phillips' equations and both actuator disk models at the same Mach number, but with different advance ratio (J=2.7) and at a lower Mach number of 0.2 (J=1.4). From these simulations the same conclusions as above could be drawn.
Therefore one can state that the uniform and nonuniform actuator disk model give a good representation of the steady flow field downstream of a propeller. However the non-uniform UDF is more complex than the one of the uniform actuator disk model and is computationally more expensive. Hence the uniform actuator disk model will be used as the numerical model to study propeller-wing interaction effects.

Part III: Propeller-wing interaction study

This part of the thesis will investigate the propeller-wing interaction effects numerically by making use of the numerical model that was developed in part II.

Chapter 5

Investigation of propeller-wing interaction effects

This chapter will focus on the propeller-wing interaction study and will especially investigate the influence of the rotation sense of the propellers on the resulting flow field. To this end four different simulations were performed:

1. Propellers off
2. Propellers on, both rotating inboard up
3. Propellers on, both rotating outboard up
4. Propellers on, down-between-engine concept

First a short discussion will be given concerning the creation of the geometry and mesh. The second section will treat the Fluent settings that were applied for the propeller-wing interaction study. The third section will provide several illustrations and graphs that characterize the flow field for each above-mentioned case. This chapter will conclude with the most important observations and remarks concerning the effect of rotation sense of the propellers on the propeller-wing interference effects.

5.1 Geometry and mesh creation of the aircraft model

This section will elaborate on the generation of the geometry of the aircraft model used in the propeller-wing interaction study and will discuss the creation of its mesh. As was the case with the validation process (Chapter 4) the program Gambit was used to create the geometry and the surface mesh. However, the volume mesh was generated by making use of the program T-Grid.

5.1.1 Creation of the geometry

The geometry used in the simulations is a generic aircraft and consists of:

- A wing
- 2 wing-mounted propeller engines (including nacelles)
- A fuselage

No tail surfaces are included in this geometry since the primary goal of this thesis is the study of the importance of the rotation sense of the propellers on the propeller-wing interference effects. No attention is paid to the effect of the slipstream on the stability and control characteristics of the airplane.
Since the tail surfaces are excluded and the geometry of the model is symmetric with respect to the xz-plane, only half of the airplane geometry needs to be modelled. This will greatly reduce the amount of cells that are needed to mesh the geometry, leading to a less expensive grid. Hence the calculation time and resources are reduced or one can opt for a more dense mesh. The set-up of the geometry is based on the geometry used in the master thesis of Tom Mullier (Mullier [2009]). The wing was created by Tom Mullier by making use of 2 different airfoil geometries (NACA 64a410 & NACA 64a380). Part of the wing extending from root to tip has a constant chord and makes use of the same airfoil geometry. At a certain point (y/y_{ref}=0.27) the trailing edge of the wing varies linearly towards the tip of the wing. A representation of the wing can be found in Figure 5.1 while Table 5.1 contains information regarding the wing span and the root, tip and mean chord length. In contrast to the geometry drawn by Tom Mullier, who placed the wing at a zero angle of attack, an angle of attack of 2° is used in the current geometry.

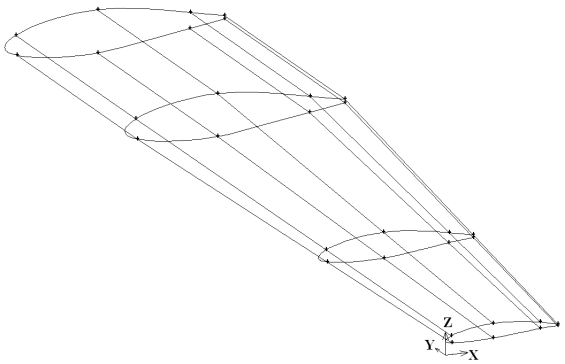


Figure 5.1: Wing geometry

The nacelles of the engines were extracted from a CAD drawing and "cleaned" for importation in Gambit by Tom Mullier. Although the same geometry of the nacelles were used, the placement of the nacelles differs. The coordinates of the hub point of the inboard and outboard engine can be found in Table 5.2. From this table it can be seen that the outboard nacelle has a larger z-coordinate than the inboard nacelle, hence it is located at a higher position with respect to the leading edge. This can also be seen by inspecting Figure 5.2.

Table 5.1: Characteristics of the wing

Parameter	Value
wing span, b	40.41 m
root chord, c_{root}	4.88 m
tip chord, c_{tip}	2.37 m
mean chord $\bar{c}$	3.73 m
Wing surface area, S_{wing}	75.29 m^2

Table 5.2: Hub point coordinates

Engine	Hub point coodinates
Inboard engine	(-3.3, 15.205, -0.28)
Outboard engine	(-3.3, 10.205, -0.12)

The fuselage is represented by a simple cylinder with a diameter of 4 m extending from the inlet to the outlet of the domain. The calculation domain consists of a rectangular block with a length of 21.32 m, a height of 16.44 m and a width of 22,595 m. A representation of the airplane geometry and calculation domain can be found in Figure 5.2.

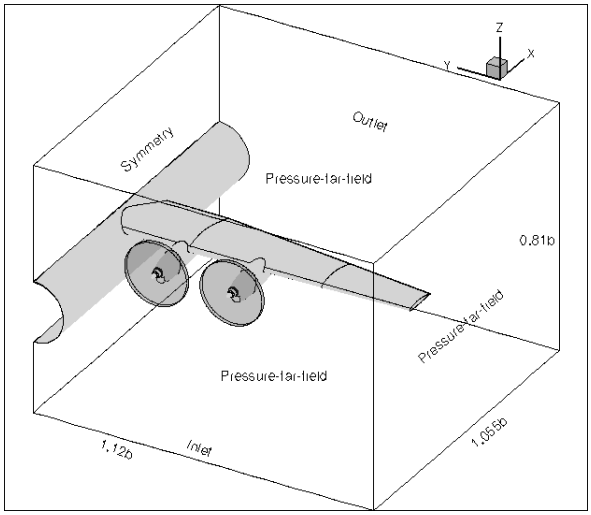


Figure 5.2: Complete airplane geometry

5.1.2 Generation of the mesh

The mesh of the propeller-wing geometry is created by making use of the programs Gambit and T-Grid. Gambit is used to create the surface mesh of the aircraft geometry, the boundaries of the calculation domain and the volume mesh of the propeller disks. T-Grid

is used to fill up the rest of the domain.
After the creation of the mesh, it will be locally refined in the region between the propeller and the TE of the wing. Therefore this section is divided into three subsections: the first subsection will discuss the mesh generation using Gambit while the second one will focus on the volume mesh generation by making use of T-Grid. The last subsection will shortly explain how the mesh is refined by making use of the "Adapt" option in FLUENT.

Mesh generation using Gambit

From the the previous section it followed that the propeller-wing geometry consists of 3 parts: the fuselage, the wing and both engines. The engines consist of a spinner-nacelle geometry and the propeller disk. From the discussion of the UDF set-up in section 4.2 it followed that there are specific requirements for the mesh of the propeller in order for the UDF to work properly. As was the case with the validation simulations, the volume beaten by the propeller will be split into two separate volumes:

1. Propeller volume
2. Actuator volume

As required by the UDF, the actuator volume consists of a depth of one cell. These cells have a width of 0.05 m in order to have a good resolution while keeping the maximum total amount of cells to a minimum. At the moment of the mesh creation the RMA RMA was limited to a maximum amount of cells of $7 \cdot 10^6$ cells due to a limited amount of memory and licenses. The current limit of the maximum amount of cells is around $20 \cdot 10^6$ since the memory problem has been solved due to the purchase of a "super computer". However the limited amount of licenses is still limiting the maximum amount of cells.
There are 53 cells in radial direction due to the applied size function. The cells' starting size and growth rate of this size function is equal to the one used to create the boundary layer on the aircraft's surface (see next subsection) in order to have a smooth transition from the nacelle's hub to the volumes beaten by the propeller (see Figure 5.3). The maximum cell volume for the size function in radial direction is equal to the depth of the actuator volume. Since it is impossible to mesh a complete circle with a radial size function, the volumes beaten by the propeller consist of four one-quarter cylinders, enabling the use of a size function in radial direction. There are 260 cells in tangential direction, leading to a tangential cell size of 0.05 m at the tip of the blade.

Again an inflow and outflow volume is added as was the case with the SR-1 mesh. The reason for the inclusion of these volumes is to ensure that the solution is not influenced by the transition from tetrahedrons to hexahedrons. Since only the inflow, propeller, actuator and outflow volumes will be meshed with hexahedrons (necessary for the UDF) while all the other surfaces and volumes will be meshed with tetrahedrons. The use of hexahedrons in this complex geometry is very difficult, and more cells would be needed to fill up the calculation domain.
Two size functions are imposed in axial direction: one at the inflow volume and one at the propeller volume. These size functions are applied to ensure a smooth transitions from

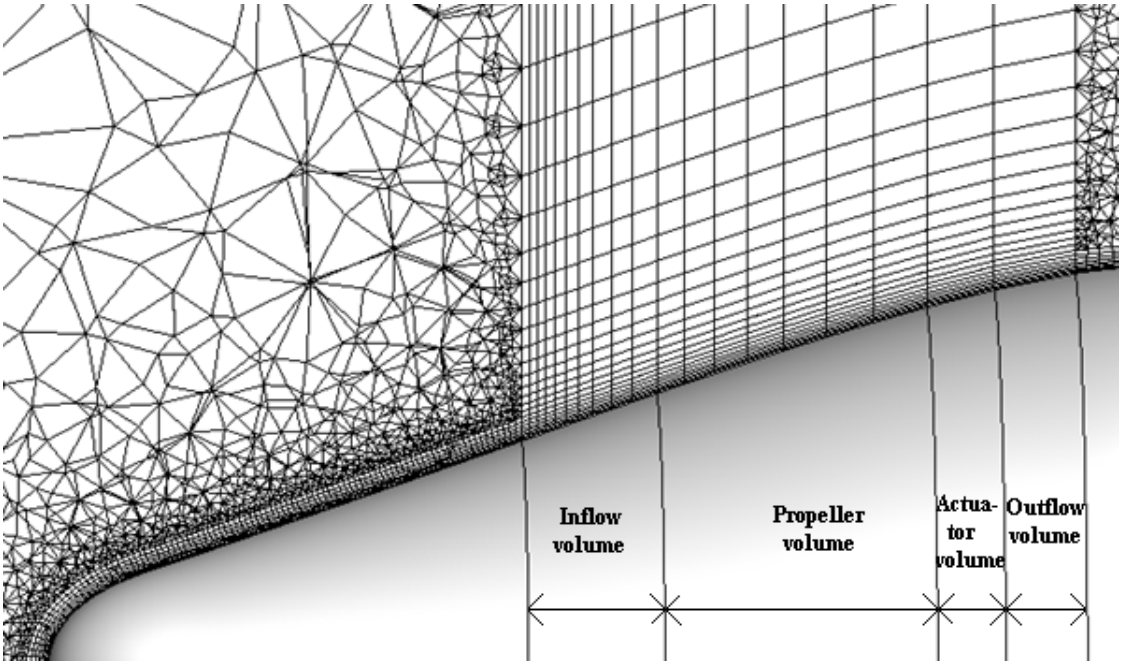


Figure 5.3: Zoom of transition of mesh from hub to propeller

the small cell size on the hub to the required axial cell size of 0.05 m inside the actuator volume (see Figure 5.4). Both size functions have a growth rate of 1.2 and a maximum size of 0.05 m however the starting size differs. The starting size of the inflow volume size function is 0.004 m (which corresponds with the size at the hub) while the starting size of the propeller volume size function is the size of the largest cell of the inflow volume. This leads to 7 columns of cells in axial direction for the inflow volume and 10 columns of cells for the propeller volume. Since the depth of the outflow volume is just 0.06 m, only one column of cells is used in axial direction.

While applying the UDF for the first time, it was discovered that the nacelle of this geometry is not 100% axisymmetric. Although the error is only of the order of 0.001 m, it has a profound effect on the proper function of the UDF because of the boundary layer which has a starting size (first height of the boundary layer) of 0.001 m (see section 5.1.2). Hence the original UDF was rewritten as was already mentioned in section 4.2.

The surface mesh of all surfaces, except the surfaces belonging to the propeller, are meshed with triangular elements. The nacelle's mesh uses a very fine mesh for the hub cone (0.0041 m interval size). A size function is applied to that part of the nacelle that is located after the outflow volume of the propeller, such that the cells will grow to a maximum interval cell size of 0.05 m. This is done to ensure a smooth transition from nacelle to wing surfaces which will also consist of triangular cell elements with an interval size 0.05 m, see Figure 5.5. The hub's mesh of the outboard engine is depicted in Figure 5.6.

A curvature size function is applied to the LE of the wing to grow cells with an interval size of 0.01 m to 0.05 m. Also a size function is applied at the wing's upper and lower surfaces to decrease from 0.05 m to an interval size of 0.015 m since this is the interval

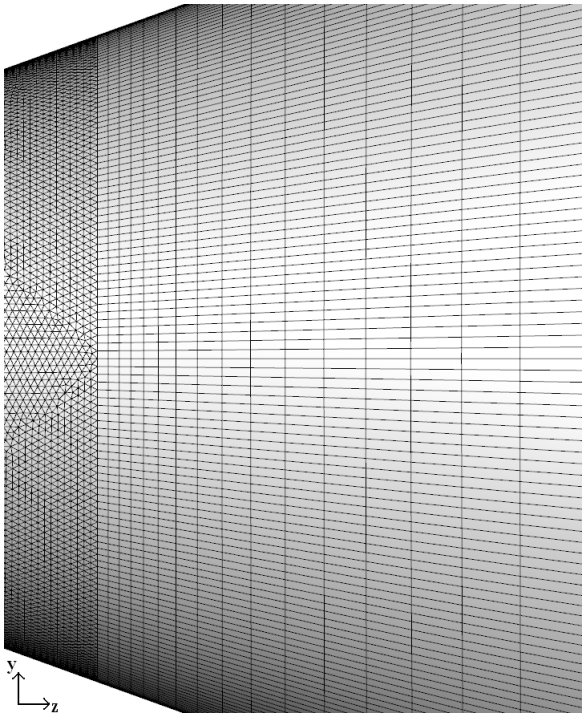


Figure 5.4: Zoom of transition of the surface mesh from hub to propeller

size that is used to fill up the TE face (see Figure 5.7). Although the behaviour of the flow near and at the fuselage is not of primary concern, a meshed size function is created with the intersection edge of the wing and the fuselage as source, a growth rate of 1.2 and a maximum cell size of 0.5 m. This meshed size function is applied to ensure that the solution near the inboard propeller is not affected due to a too crude meshing of the fuselage. The symmetry boundary faces above and below the fuselage are also subjected to a size function with a starting size of 0.5 m, a growth rate of 1.1 and a maximum cell size of 1 m in order to minimize the number of cells. The mesh of the symmetry boundary can be found in Figure 5.8.

Also size functions are applied to the other boundaries, leading to a maximum cell size of 1 m to minimize calculation costs. Since the volume mesh around the aircraft will be generated from the cells on the aircraft's surface the larger cell sizes on the boundaries will not affect the volume cells close to the airplane's wing and engines. Furthermore the mesh behind the propellers will be refined in cylinders that extend till just downstream of the TE of the wing, and have a slightly larger diameter than the propeller (see the last subsection).

Mesh generation using T-Grid

After the creation of all surface meshes and the volume mesh of the propellers the volume mesh of the remaining domain will be generated by making use of T-grid. The first step is the generation of the boundary layer of the aircraft. The creation of a boundary layer is necessary to correctly simulate the turbulent behaviour near the wall, especially when using the $k-\epsilon$ and $k-\omega$ turbulent models (see section 3.2.1). These models were designed

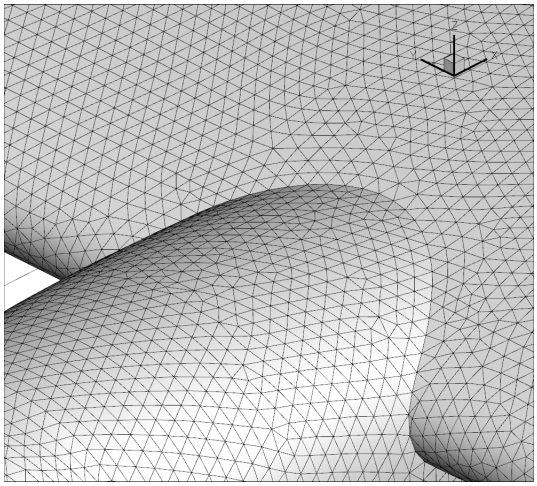


Figure 5.5: Zoom of transition of mesh from nacelle to wing

for freestream flows, located far away from walls. Therefore the behaviour near the wall can not be resolved but use is made of a wall function to simulate its behaviour close to the wall. The parameter y^+ is a parameter which states whether the boundary layer is suitable to resolve turbulent flow. The formula for the determination of y^+ was already given in Eq (3.17), but it is repeated below for convenience:

$$y^+ = \frac{c_\mu^{\frac{1}{4}} \sqrt{k}\, y_P}{\nu} \tag{5.1}$$

The term y_P denotes the height of the first cell. Since the friction velocity can be expressed with formula (5.2) and the wall shear stress with Eq (5.3), the friction velocity, $u_\tau = c_\mu^{\frac{1}{4}} \sqrt{k}$ can also be rewritten as Eq (5.4).

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{5.2}$$

$$\tau_w = c_f \frac{1}{2} \rho V_\infty^2 \tag{5.3}$$

$$u_\tau = V_\infty \sqrt{\frac{c_f}{2}} \tag{5.4}$$

The term c_f is the skin friction coefficient and can for a turbulent boundary layer on a flat plate according to Prandtl's Power law (White [2006]) be predicted by Eq (5.5).

$$c_f = 0.058 Re^{\frac{-1}{5}} \tag{5.5}$$

Substituting Eq (5.4) into Eq (5.1) the expression for y^+ becomes:

$$y^+ = \frac{Re_{\bar{c}} \sqrt{\frac{c_f}{2}}\, y_P}{\bar{c}} \tag{5.6}$$

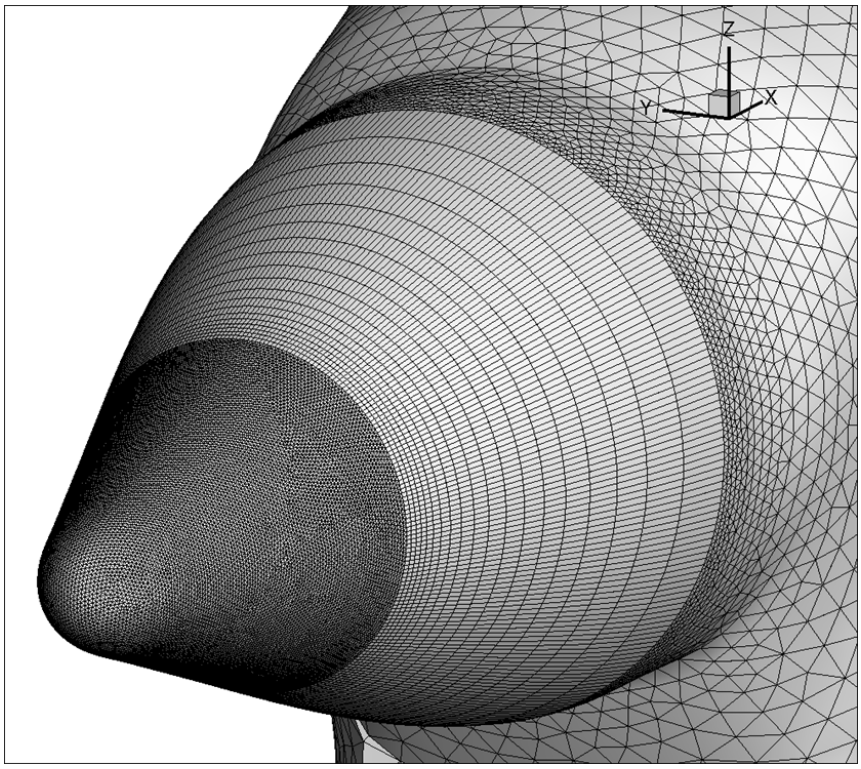


Figure 5.6: Zoom of transition of mesh from hub to propeller

The term $\bar{c}$ is the average chord of the wing.

Solving Eq (5.6) for the first cell's height, y_P leads to formula (5.7):

$$y_P = \frac{y^+ \bar{c}}{Re_{\bar{c}} \sqrt{\frac{c_f}{2}}} \tag{5.7}$$

Since FLUENT stores flow variables in the centroid of a cell, Eq (5.7) needs to be doubled in order to have the correct estimation of the height of the first cell:

$$h_{1\text{st cell}} = \frac{2y^+ \bar{c}}{Re_{\bar{c}} \sqrt{\frac{c_f}{2}}} \tag{5.8}$$

From the previous discussion in section 3.2.1 it followed that it is desirable that $y^+ <= 100$. Taking y^+ equal to 80 and filling in the flight parameters from Table 5.5, leads to $h_{1\text{st cell}} = 0.001$.

Hence the boundary layer will have a starting size of 0.001 m, with a growth factor of 1.2 and will consist of 8 layers.

In order to have a smooth transition from hexahedrons to tetrahedrons a layer of pyramid cells is "glued" to all rectangular mesh faces of the outer boundaries of the propeller disks. To fill up the rest of the domain the "auto mesh" function of T-Grid is used which will grow cells from the airplane's surface, making use of the surface mesh, towards the boundaries. In order to minimize calculation cost, the growth rate is chosen to be 1.4. This results in a total number of cells of 6 012 070.

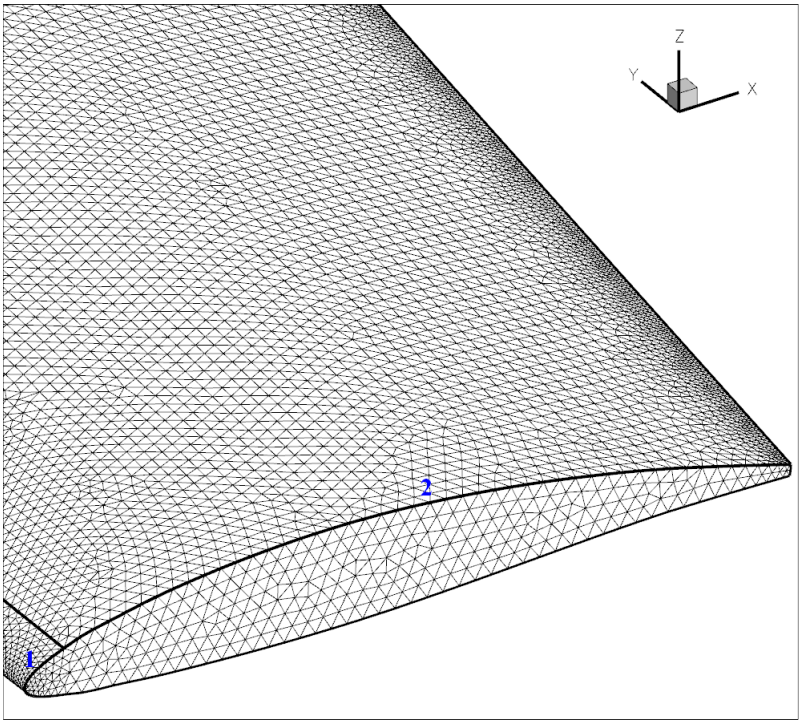


Figure 5.7: Size functions applied to the wing

Local mesh refinement

After importation of the mesh file in FLUENT, the grid is modified by using the "Adapt" command. The grid will not be adapted everywhere in the calculation domain, hence the "region" option is selected. Only the cells are changed that are enclosed by the cylinder which has a radius of 2.15 m and an axis of rotation specified by a starting and end point of which the coordinates are given in Table 5.3. Hence the cylinders extend from the outflowface of the outflow volume to about 1.46 m behind the TE of the wing.

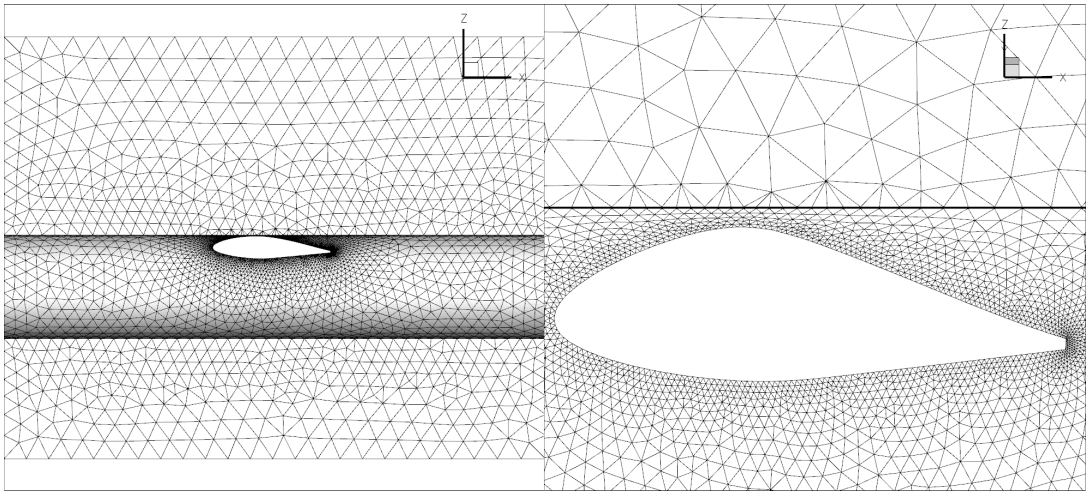
Table 5.3: Coordinates specifying the axis of rotation of the refinement cylinder

Engine	Coordinates of the starting point	Coordinates of the end point
Inboard engine	(-2.589, 15.205, -0.20)	(6, 15.205, -0.20)
Outboard engine	(-2.589, 10.205, -0.12)	(6, 10.205, -0.12)

The control parameters for the refinement are summarized in Table 5.4. The minimum cell volume is calculated by means of Eq (5.9), which is the formula to calculate the volume of a tetrahedron based on the edge length a:

$$\text{Vol} = \frac{\sqrt{2}}{12}\, a^3 \tag{5.9}$$

where $a = 0.05\ m$.



(a) Symmetry boundary and fuselage mesh

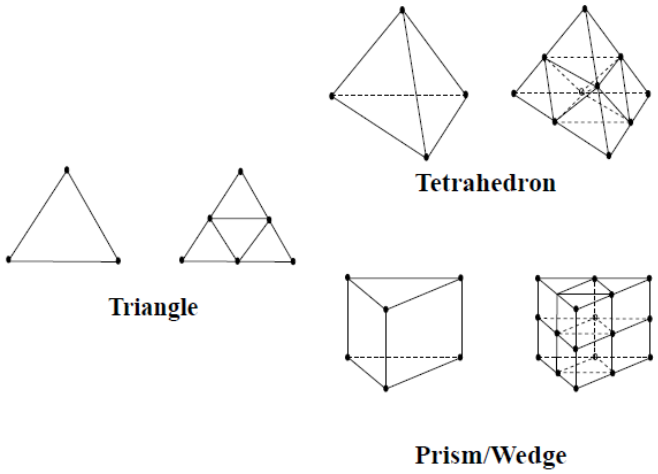
(b) Zoom-in of fuselage mesh

Figure 5.8: Symmetry boundary and fuselage mesh

Table 5.4: Refinement controls

Parameter	**Outboard Engine**	**Inboard Engine**
Minimum cell volume	$1.47 \cdot 10^{-5} m^3$	$1.47 \cdot 10^{-5} m^3$
Minimum number of cells	0	0
Maximum number of cells	9 012 070	12 012 070
Maximum level of refine	2	2
Volume weight	1	1

FLUENT adapts the mesh according to the "hanging node principle", which is depicted in Figure 5.9. From Figure 5.9 it can be seen that the "hanging node principle" splits one large triangular 2D element into 4 smaller triangles. For a 3D triangular element: one large tetrahedron is divided into 8 smaller tetrahedra, while a prism is split into 8 smaller prisms. After the mesh refinement has been applied, the grid consists of 10 683 076 cells in total.



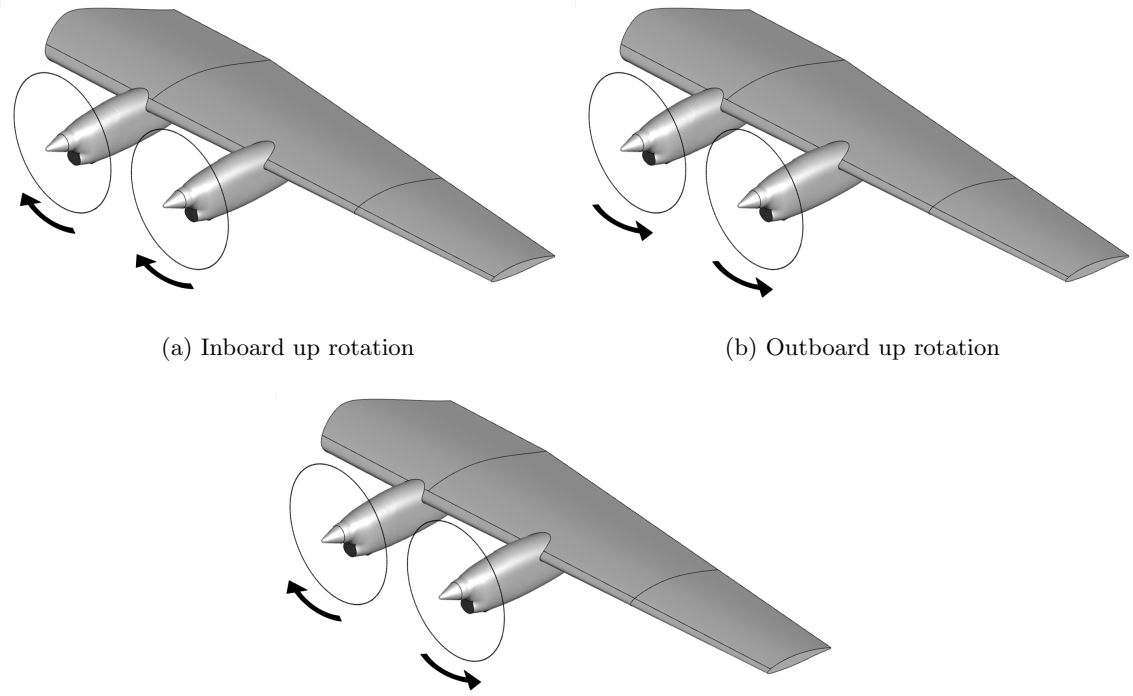
(a) 2D (b) 3D

Figure 5.9: The hanging node adaption principle "ANSYS FLUENT 12.0 Theory Guide" [2009]

5.2 Flight conditions and cases

This section will discuss the cases that will be treated in this thesis. The reference and most simple case is the prop off case in which the engines are running but the propellers are not producing thrust, hence no induced velocity is imposed. The influence of the rotation sense will then be investigated by rotating the propellers as in Figure 5.10:

- Inboard up
- Outboard up
- Down-Between-Engine (DBE)



(a) Inboard up rotation

(b) Outboard up rotation

(c) DBE rotation

Figure 5.10: Rotation cases

The flight parameters are stated in Table 5.5 and the parameters for the engines at the stated flight altitude and speed are summarized in Table 5.6.

Table 5.5: Flight conditions

Parameter	Value
Flight altitude	35 000 feet or 10 668 m
Freestream Mach number M_∞	0.6
Static (operating) pressure	23840 Pa
Static temperature	218.81 K
Density	0.3796 kg/m^3
Freestream velocity, V_∞	177.91 m/s

Table 5.6: Engine parameters

Parameter	Value
Air flow intake	6.5 kg/s
Mass flow oil cooling	1 kg/s
ESFC	0.5 kg/(W s)
Mass flow exhaust	8 kg/s
Jet total temperature	824.3 K
Propeller total thrust	8489.69 N
Propeller rotational speed	17.17 rps
Propeller diameter	4.11 m

5.3 Fluent settings

This section will shortly discuss which FLUENT settings were applied in the propeller-wing interaction study.

5.3.1 Model settings

A 3-dimensional, steady, compressible, turbulent flow without heat transfer will be modelled. The "Realizable $k-\varepsilon$" method is chosen as turbulent model with "Non-equilibrium wall functions" and standard parameters.

5.3.2 Material properties

The air is considered as an ideal gas. The freestream Mach number is set to 0.6, which corresponds to a Reynolds number of $14 \cdot 10^6$.
The nacelle material is set to aluminum and the default values are used.

5.3.3 Solver controls

As stated previously, the pressure-based solver is used since the density-based solver does not allow the attachment of a UDF in order to impose the velocities in the actuator volume. For more information concerning these solvers, the reader is referred to section 3.2.3. The SIMPLE algorithm is chosen for the pressure-velocity coupling.
There are different discretization schemes available for the pressure, density, momentum, energy, turbulent kinetic energy and turbulent dissipation rate. To initialize the solution, the first order discretization method is used for all parameters. Afterwards the second order discretization scheme is selected. The gradient terms are determined by means of the Green-Gauss node-based calculation method since this method is more accurate than cell-based gradient calculation when an unstructured mesh is used.

5.3.4 Boundary conditions

The top, bottom and tip faces of the domain are categorized as a pressure-far-field boundary condition, which is used to model freestream conditions at infinity. The freestream Mach number and direction must be given as input as well as the freestream static temperature. A symmetry boundary condition is applied to the root face, see Figure 5.2. This boundary condition allows to model only half of the airplane geometry, which greatly reduces the total amount of cells of the mesh. The inlet is modelled as a pressure-inlet on which a certain gauge total pressure is employed, while the outlet is modelled as a pressure outlet, with zero gauge static pressure. This difference in gauge pressure between the inlet and outlet will produce a flow of prescribed Mach number. From section 3.2.4 it followed that Eq (3.33) is used to obtain the required total gauge pressure at the inlet and

Eq (3.34) to find the corresponding total temperature. For convenience both formulae are repeated below:

$$p_{t_{gauge}} = p_{op}\left(1 + \frac{\gamma - 1}{2}M^2\right)^{\frac{\gamma}{\gamma-1}} - p_{op} \tag{3.33}$$

$$\frac{T_t}{T_s} = \left(1 + \frac{\gamma - 1}{2}M_\infty^2\right) \tag{3.34}$$

Filling in the parameters from Table 5.5 and $\gamma = 1.4$ leads to:
$p_{t_{gauge}} = 6800\ Pa$ & $T_t = 234.56\ K$

All surfaces related to the airplane's geometry are modelled as adiabatic no slip walls, except for the surfaces related to the actuators, which are set to interior faces. Since the flow is turbulent, turbulence parameters need to be specified for all boundaries of the domain. The governing turbulence parameters are the turbulent intensity and length scale. The length scale was obtained by Eq (5.10):

$$l = 0.07 \cdot L_c \tag{5.10}$$

The term L_c is the characteristic length scale of the object. Since we also want to take the smallest characteristic length scale into account $L_c = c_{tip} = 2.37\ m$. The turbulent length scale is then equal to 0.17 m.
The turbulence intensity is set to 1% for all boundaries except for the mass flow inlet boundary conditions which are applied at the engines. Table 5.7 gives an overview of the turbulence parameters for each boundary.

Table 5.7: Turbulence parameters

Boundary	**Turbulence intensity**
Oil cooling inboard & outboard engine	1.5%
Intake inboard & outboard engine	1.5%
Exhaust inboard & outboard engine	2%
Top, bottom and tip domain (pressure-far-field BC)	1%

5.4 Set-up of the simulation and iterative convergence

The simulations can not be performed by immediately imposing the Fluent settings as described in section 5.3, since these conditions are too demanding for the FLUENT solver. A way around this problem is to start up the simulation with very simple settings e.g. M=0.2 & laminar flow. Let this simulation iterate until the residuals show a nice decreasing or steady behaviour and save the obtained data. Then alter the settings e.g. higher Mach number, load the previous data and let it iterate again as before. Repeat this process until the desired settings are obtained and the required freestream Mach number is

found at the inlet of the calculation domain. In total 7800 iterations were performed to find a uniform flow which complies with all boundary conditions and flight parameters for the prop off case. Then the mesh was locally refined in cylinders behind the propellers, as was discussed in section 5.1.2. Another 2000 iterations were performed without imposing the UDF (prop off case). The case and data file which were obtained by this simulation is the start-up file for all the other cases. To ensure that the simulation is converged for all cases, 6000 iterations are performed when the UDF is imposed. For one of the cases the overall lift and drag coefficient was written to a text file. From this file it could be seen that the overall lift coefficient did not change anymore for 3 significant digits for the last 2000 iterations. The overall drag coefficient did not change for 4 significant digits for the last 1500 iterations. This is a clear indication that the solution is converged since these 2 important parameters remain constant. Normally the residuals of the simulation also give a good indication of the convergence of the solution if they are small enough and don't fluctuate much. However they can also be misleading and it is therefore necessary to monitor whether import parameters such as C_L and C_D remain constant. In order to compare the prop on cases with the prop off case, also 6000 extra iterations were performed for the prop off case. This was done to ensure iterative convergence.

5.5 Results of the propeller-wing interaction study

This section will discuss the results of the propeller-wing interaction study. The main interest of this study is the influence of the propeller on the wing and how the rotation sense influences the spanwise distribution of the lift and drag of the wing. This section is divided into three subsections: first the lift and drag characteristics for each case will be considered. Then C_p isolines and pressure coefficient plots will be given to explain the results of the first subsection. This section will conclude by specifying the main results that were obtained by comparing all cases.

5.5.1 Lift and drag characteristics of the wing

It is possible to calculate the overall lift and drag force by means of FLUENT. Since the dynamic pressure and the reference wing surface is known the overall lift and drag coefficient can be calculated by means of respectively Eq (5.11) and Eq (5.12):

$$C_L = \frac{L}{\frac{1}{2}\rho V_\infty^2 S_{wing}} \tag{5.11}$$

$$C_D = \frac{D}{\frac{1}{2}\rho V_\infty^2 S_{wing}} \tag{5.12}$$

The lift and drag forces obtained by FLUENT for all cases and the corresponding lift and drag coefficients[1] are summarized in Table 5.8

[1]The lift and drag is only calculated for the wing and nacelles, the effect of the fuselage is not taken into account.

Table 5.8: Lift and drag parameters of the wing as obtained by Fluent

Case	L [N]	D [N]	C_L [-]	C_D [-]	ΔC_L [-]	ΔC_D [-]
Prop off	182698	11420	0.404	0.0253	-	-
Inboard up	192195	14961	0.425	0.0331	+5%	+30.8%
Outboard up	184602	15060	0.408	0.0333	+1%	+31.6%
DBE	185584	14852	0.410	0.0328	+1.5%	+29.6%

The inboard up case appears to deliver the highest overall lift coefficient according to Table 5.8. By letting both propellers rotate outboard up only a slight increase in lift coefficient w.r.t. the prop off case is obtained while it produces the highest overall drag coefficient, leading to the worst aerodynamic performance of all three cases. The DBE case has one inboard up and one outboard up rotating propeller, which leads to a small augmentation in overall lift coefficient which is larger than the outboard up case, but smaller than the inboard up case. The overall drag coefficient is however the smallest of all three cases.

In order to know the spanwise distribution of the lift and drag coefficients a Matlab m-file was created which calculates the local lift and drag coefficient by chordwise integration of the pressure and friction forces. To this end 51 airfoil sections were extracted from the wing. For each section the static pressure and friction forces were extracted. The sectional lift and drag coefficient are based on the local chord of the wing. The local lift and drag coefficients can be determined with formulae (5.13) and (5.14):

$$C_l = \frac{l}{\frac{1}{2}\rho V_\infty^2 c} \tag{5.13}$$

$$C_d = \frac{d}{\frac{1}{2}\rho V_\infty^2 c} \tag{5.14}$$

where l and d are the sectional lift and drag force respectively.

The m-file which calculates the local lift and drag coefficient can be found in Appendix C. Appendix D contains the journal files that were used to extract the airfoils and their data. The resulting spanwise distribution of the lift and drag coefficients can be found in Figure 5.11 and Figure 5.12, respectively. It should be noted that the wing sections which intersect the nacelles are not included in the figures. The full black verticals in Figure 5.11 and Figure 5.12 mark the outlines of the nacelles, while the dashed verticals indicate the outerlines of the propellers.

From Figure 5.11 it can be seen that inboard up rotation of the propellers produces the highest lift in contrast to the wing with outboard up rotating propellers which produces the lowest lift. This could also be derived from Table 5.8. Furthermore it can be seen that the spanwise lift distribution of the DBE case follows the same trend as the inboard up rotating propellers, however with a small offset, until it is in between both propellers.

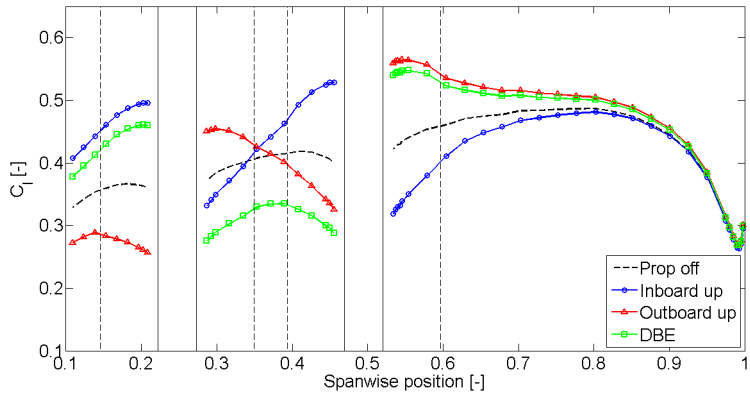


Figure 5.11: Spanwise lift coefficient ("—" nacelle borders, "- -" propeller diameter)

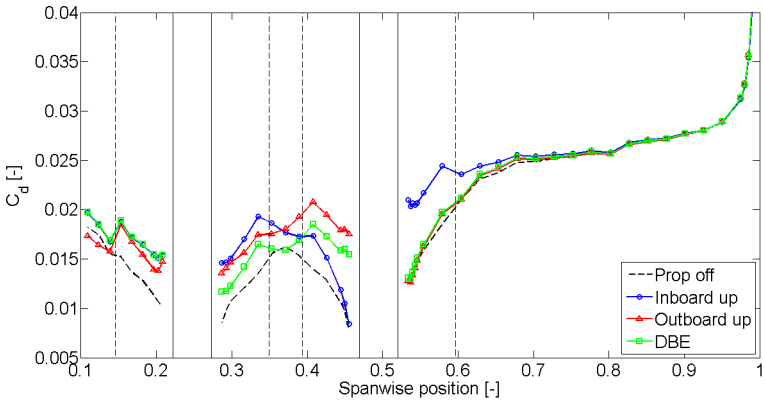


Figure 5.12: Spanwise drag coefficient ("—" nacelle borders, "- -" propeller diameter)

There the lift distribution decreases and starts following the same trend as the lift distribution of the outboard up case. The offset in lift coefficient between the DBE and outboard up rotating case is smaller than the one observed for the inboard up and DBE rotation. The larger offset is probably caused by the mutual interaction of both inboard up rotating propellers, which has a favourable effect on the lift distribution. For the DBE case this favourable interaction disappears resulting in lower C_l values when compared to the inboard up case. Also the proximity of the fuselage plays a role in this matter as the outboard engine is less influenced. At 95% of the span the C_l-line of all cases coincide. At this span position the propeller slipstream no longer influences the wing lift distribution. By comparing Figure 5.11 with Figure 2.9 from section 2.1.4, one can see that the numerical prediction of the spanwise lift coefficient agrees well with the shape and behaviour predicted by theory. Due to the spanwise gradients of the wing, the change in lift coefficient distribution caused by the propeller slipstream is not limited to the wing located inside the slipstream. Therefore for an inboard up rotating propeller the local C_l at the root is larger than the prop off case. The opposite is true for an outboard up rotating propeller which has a lower local lift coefficient at the wing's root. The effect of increase in dynamic pressure due to the induced axial velocity generated by the propeller behaves symmetrical with respect to the thrust axis. When based on the local flow conditions, the

local lift coefficient does not change. However, based on the undisturbed conditions, there is an increase in local lift coefficient independent of rotation sense of the propeller (Figure 2.6). Furthermore the tangential velocity causes an in- or decrease in local angle of attack of the trailing wing, depending on the fact whether the wing is located in the upwash (upward going blade) or downwash (downward going blade) region of the propeller blade (Figure 2.8). The combined effect of the induced axial, tangential and spanwise wing gradient is depicted in Figure 2.9. The same behaviour is displayed in Figure 5.11.

By inspecting Figure 5.12 one can deduce that the spanwise distribution of the drag coefficient of the DBE rotation first follows the same trend as the inboard up case and then switches again to follow the trend of the outboard up rotating propellers. It is difficult to derive which case produces the least drag. From Table 5.8 it followed that the outboard up case produces the largest overall drag coefficient. But the difference in overall drag coefficient is very small.
There is no longer an offset between the inboard up rotating propeller and the DBE rotation as was observed for the lift coefficient. And also the C_d-line of the outboard up case and DBE configuration coincide when they display the same behaviour. Only in the interaction region of both propellers a small offset between the DBE and the in- and outboard up case can be observed. Furthermore the C_d-line for all configurations coincide at a spanwise position of 75%. It can therefore be concluded that the rotation sense of the propellers has a stronger influence in spanwise direction on the spanwise lift distribution than on the spanwise drag distribution.

Concerning the performance of the aircraft, the range and endurance are important parameters. Their formulae are given by respectively Eq (5.15) and Eq (5.16)(Ruijgrok [1996]).

$$R = \frac{\eta_j}{c_p} \cdot \frac{C_L}{C_D} \cdot ln \frac{W_1}{W_2} \tag{5.15}$$

$$E = \frac{\eta_j}{c_p} \cdot \sqrt{\frac{C_L^3}{C_D} \cdot \frac{\rho S_{wing}}{2}} \cdot \left(\frac{2}{\sqrt{W_2}} - \frac{2}{\sqrt{W_1}} \right) \tag{5.16}$$

The term W_1 represents the initial weight at the start of cruise, while W_2 is the weight at the end of cruise. The parameter η_j indicates the propulsive efficiency and the term c_p the specific fuel consumption. From above equations it can be seen that the term C_L/C_D or lift-to-drag ratio plays an important role in the outcome of the maximum range while the term $C_L^{3/2}/C_D$ influences the maximum endurance of the airplane. These terms are summarized in Table 5.9.

From Table 5.9 it can be deduced that the inboard up configuration delivers the best results concerning range and endurance. However the differences are small. The outboard up case will have the shortest range and endurance of all cases. The performance numbers of the DBE configuration is located somewhere in between the inboard up and outboard up case. Hence from an aerodynamic point of view, the DBE case is not the best design solution. Although it will probably give better handling characteristics in case of engine failure. It is therefore recommended that also a stability and control check is done in

Table 5.9: Lift and drag parameters of the wing as obtained by Fluent

Case	$\frac{C_L}{C_D}$ [-]	$\frac{C_L^{\frac{3}{2}}}{C_D}$ [-]
Prop off	16	10.2
Inboard up	12.8	8.4
Outboard up	12.3	7.8
DBE	12.5	8

the preliminary design phase in order to see whether the performance advantages of the inboard up case outweigh the disadvantages of the stability and control characteristics in case of engine failure. Since the lift-to-drag ratio of the inboard up case is only 2.4% larger than the DBE case, while the endurance term is 5% larger.

The lift-to-drag ratio and the term $C_L^{3/2}/C_D$ based on the pressure and stress distribution of the wing can be found in Figure 5.13 and 5.14 respectively. Again the DBE case displays first the same trend as the inboard up case and switches afterwards to follow the trend of the outboard up configuration.

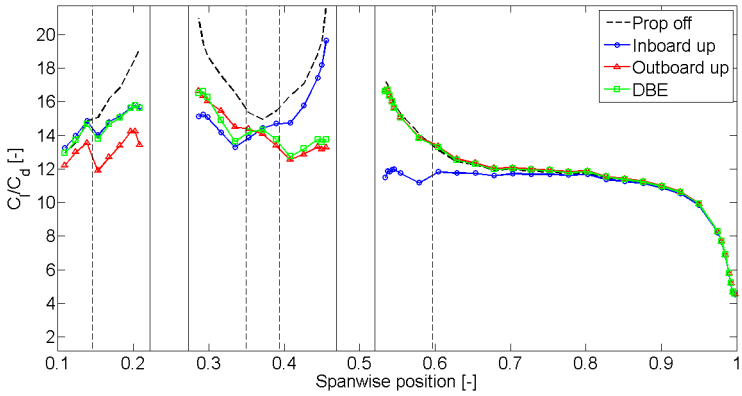


Figure 5.13: Spanwise lift-to-drag ratio ("—" nacelle borders, "- -" propeller diameter)

5.5.2 Pressure coefficients plots

In order to better understand the spanwise lift and drag distributions, pressure coefficient isolines will be given, together with pressure coefficient plots for certain spanwise locations (which are also indicated in Figure 5.15 to 5.21):

1. Close to the wing's tip, outside the influence zone of the outboard propeller (y/y_{ref}=0.901)
2. Outboard, outside the propeller diameter of the outboard engine (y/y_{ref}=0.633)
3. Outboard, inside the propeller diameter of the outboard engine (y/y_{ref}=0.546)
4. Inboard, inside the propeller diameter of the outboard engine (y/y_{ref}=0.44)

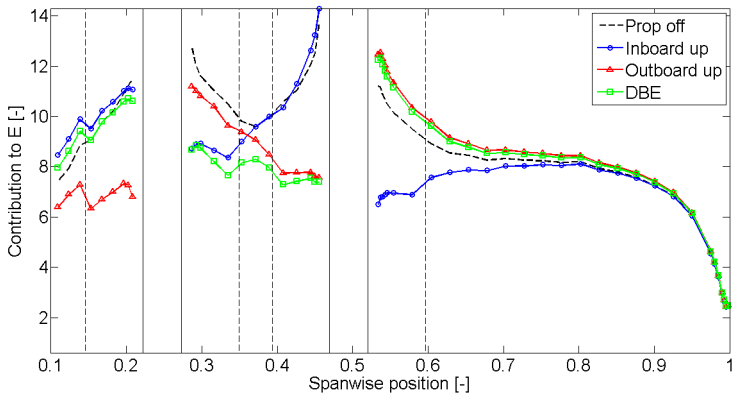


Figure 5.14: Spanwise distribution of $\frac{C_L^3}{C_D^2}$ ("–" nacelle borders, "- -" propeller diameter)

5. In between the in- and outboard engine (y/y_{ref}=0.371)
6. Outboard, inside the propeller diameter of the inboard engine (y/y_{ref}=0.297)
7. Inboard, inside the propeller diameter of the inboard engine (y/y_{ref}=0.197)
8. Inboard, outside the propeller diameter of the inboard engine (y/y_{ref}=0.109)

The pressure coefficient can be calculated by means of Eq (5.17):

$$C_p = \frac{2}{\gamma M_\infty^2}\left(\frac{p}{p_\infty} - 1\right) \tag{5.17}$$

The pressure coefficient isolines for the prop off case as well as its Cp-plots can be found in Figure 5.15.

From this figure it can be observed that there is a difference in pressure coefficient isolines for the in- and outboard nacelles. This difference is probably mainly caused by the difference in height of the engine placement. The outboard engine is placed at a slightly higher z-coordinate than the inboard engine as was already stated in the previous section. The presence of the fuselage can of course also have an influence on the pressure distribution of the inboard engine. The pressure coefficient isolines for the inboard up, outboard up and DBE cases can be found in Figure 5.17, 5.19 and 5.21 respectively. In these figures, the pressure coefficient plots are displayed with respect to the prop off case. Therefore the discussion presented below is always given with respect to the prop off case, unless mentioned otherwise.

Inboard up rotation

From C_p-plot 1 from Figure 5.17 it can be seen that the distribution of the pressure coefficient for the inboard up rotation and the prop off case is identical. This implies that the outboard propeller has no influence on this section. The same conclusion can be drawn by comparing the pressure coefficient isolines of Figure 5.17 with Figure 5.15. This also corresponds with the observation made in the previous section where it followed from Figure 5.11 that the C_l-lines of the prop off and inboard up case coincide at 90% of the span.

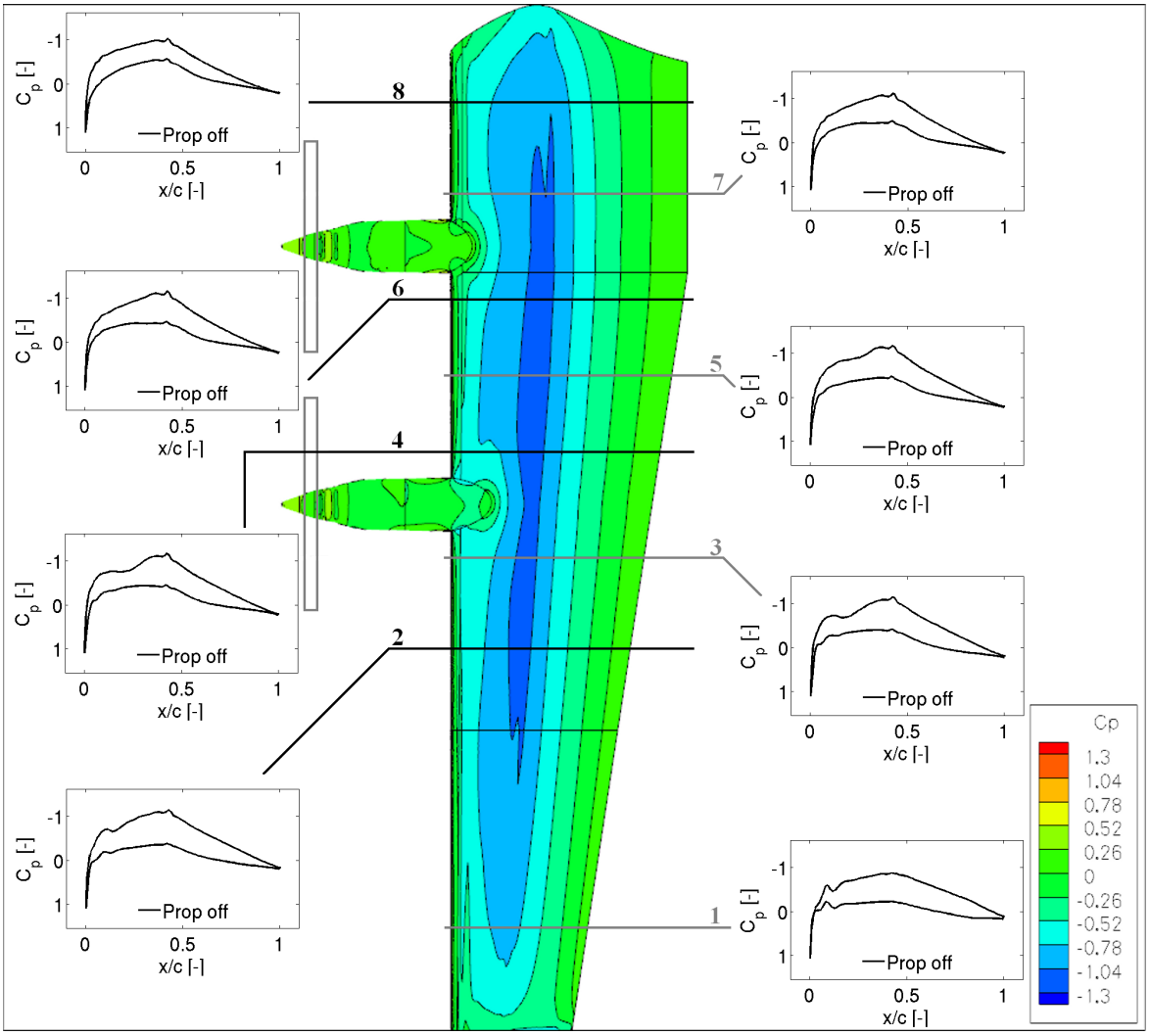


Figure 5.15: C_p plots and C_p isobars for Prop off case suction side

Although location 2 is located outside the propeller slipstream, there is a slight difference in pressure coefficient. Close to the LE C_p is slightly less negative for the suction side and slightly more negative for the pressure side. Furthermore between 35 and 64% chord there is slightly more suction on the suction side for the inboard up rotating propellers. For the pressure side, the C_p-values are slightly more negative from the LE till 70% chord. The overall effect is a decrease in overall airfoil C_l.

At section 3, a larger difference in C_p is noticed, which is not so surprising since the airfoil is located inside the slipstream of the propeller. There is a negative lift production close to the LE of the wing since the static pressure is decreased at the lower surface, leading to a more negative C_p while the pressure is augmented at the suction side. This can also be seen by comparing Figure 5.15 and 5.16 with respectively Figure 5.17 and 5.18. The C_p-values remain more negative for the pressure side until 75% chord, from here on it is slightly more positive. The pressure is augmented for the upper surface until 20% chord, where it starts to decrease w.r.t. the prop off case until 80% chord with a local peak in C_p

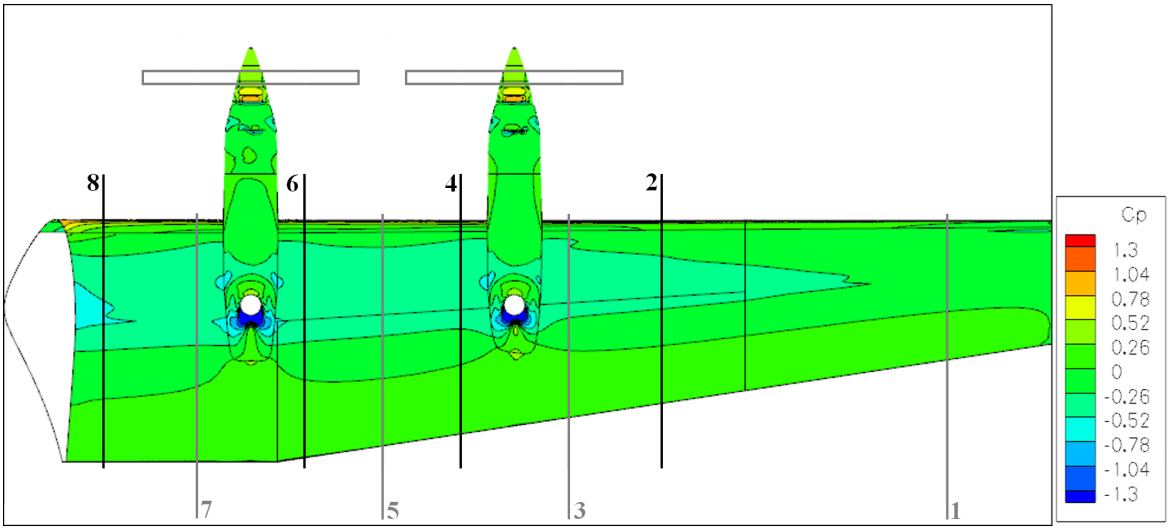


Figure 5.16: C_p plots and C_p isobars for Prop off case pressure side

at 44% chord. This behaviour leads to lower local lift coefficient, which is also confirmed by Figure 5.11

The LE suction peak is increased and the pressure coefficient is more negative for the suction side until 75% chord for section 4. While the pressure is increased until 20% chord for the pressure side, producing more positive C_p-values. From this point on the C_p-values become more negative until 65% chord, where it starts to augment again for the lower surface. The resulting local lift coefficient for this section is larger than the one obtained with the propellers off.

Right in the middle between the two engines (section 5), the pressure side is almost not affected. There is no longer an increase in LE suction, although there is a more negative C_p for the suction side from 15 till 70% chord. This leads to a higher sectional lift coefficient.

Section 6 is located inside the slipstream of the inboard propeller and more particulary in the region of the downward going blade. The behaviour of the pressure coefficient is similar to the one observed for section 3, although the negative effect on the sectional lift coefficient seems to be less pronounced. Since the production of a negative lift near the LE of the wing is less and the decrease in pressure for the suction side seems to be larger. The sectional lift will be less when compared to the prop off case, but its decrease in lift coefficient w.r.t. the prop off case will be lower than the one of the inboard case observed for section 3. This effect is probably caused by the interaction of the two rotating propellers.

Just outboard of the inboard engine (section 7) the increase in LE suction peak with respect to the prop off case is lower than the one observed for section 4 while the behaviour of the C_p values of the pressure side is similar. Furthermore the propeller slipstream seems to diminish the static pressure at the suction side more gradual and less strong than at section 4. Because of these changes in pressure distribution with respect to the prop off case, the sectional lift coefficient has a much higher value, but lower than the one of section 4 for the inboard up case.

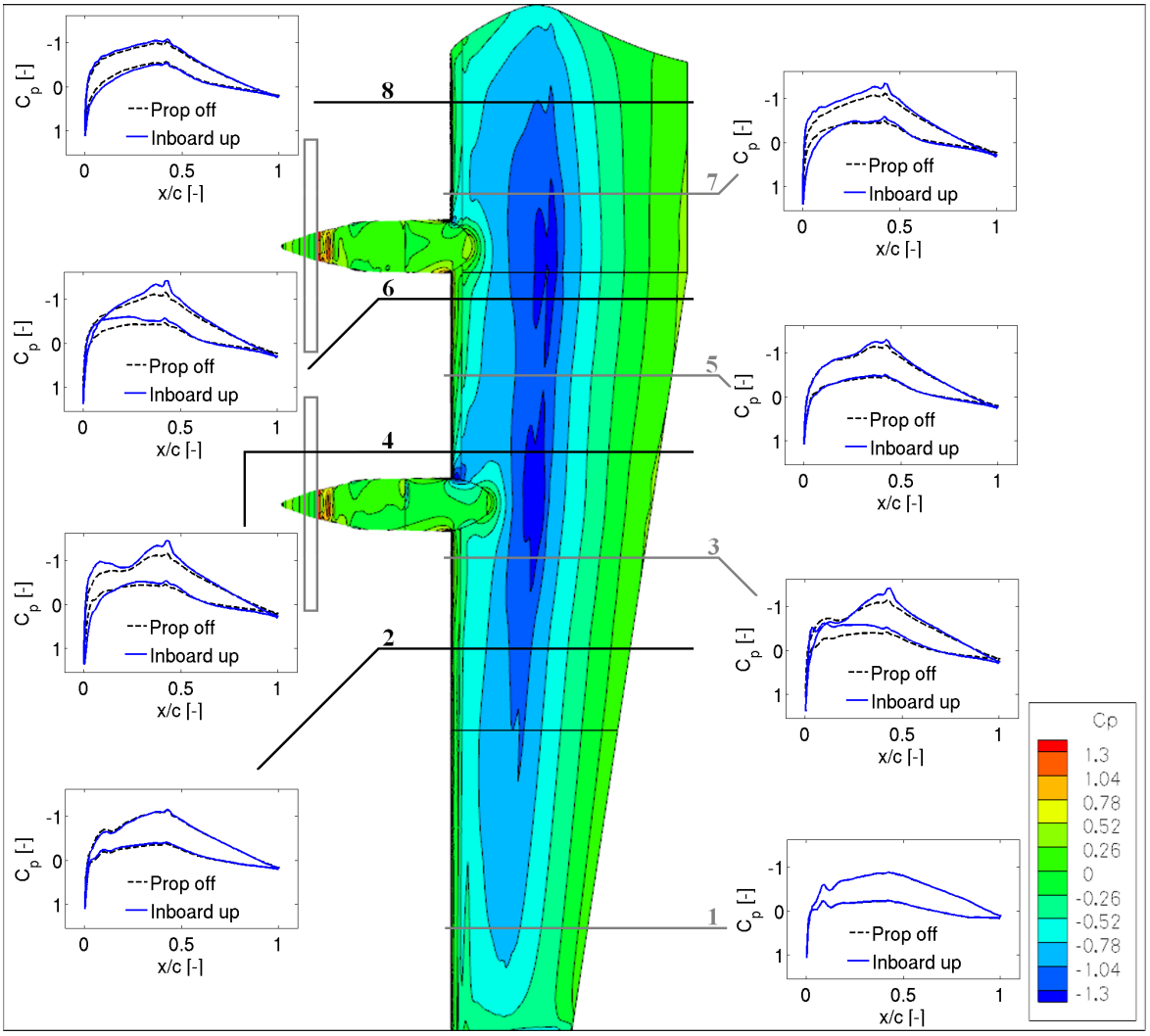


Figure 5.17: C_p plots and C_p isobars for Inboard-up rotating props suction side

Although section 8 is located outside the diameter of the inboard propeller, the influence of the slipstream is still visible. This influence, however small, translates into a slightly more negative C_p for the suction side and a slightly more positive C_p for the pressure side. Hence the sectional lift coefficient is higher.

In summary, sections 4, 5, 7 and 8 produce a larger lift coefficient when compared to the prop off case. This is the result of a more favourable pressure coefficient distribution as well as higher local velocities. Furthermore the sectional lift coefficient is unaltered for section 1 and diminished for sections 2, 3 and 6. Apparently the suction side of the wing is more affected by the propeller slipstream than the pressure side for the sections washed by the upward going blade while the opposite is true for the sections located behind the downward going blade

Outboard up rotation

By inspection of C_p-plot 1 from Figure 5.19 it can be seen that the distribution of the pressure coefficient for the airfoil of section 1 for the outboard up rotation is almost

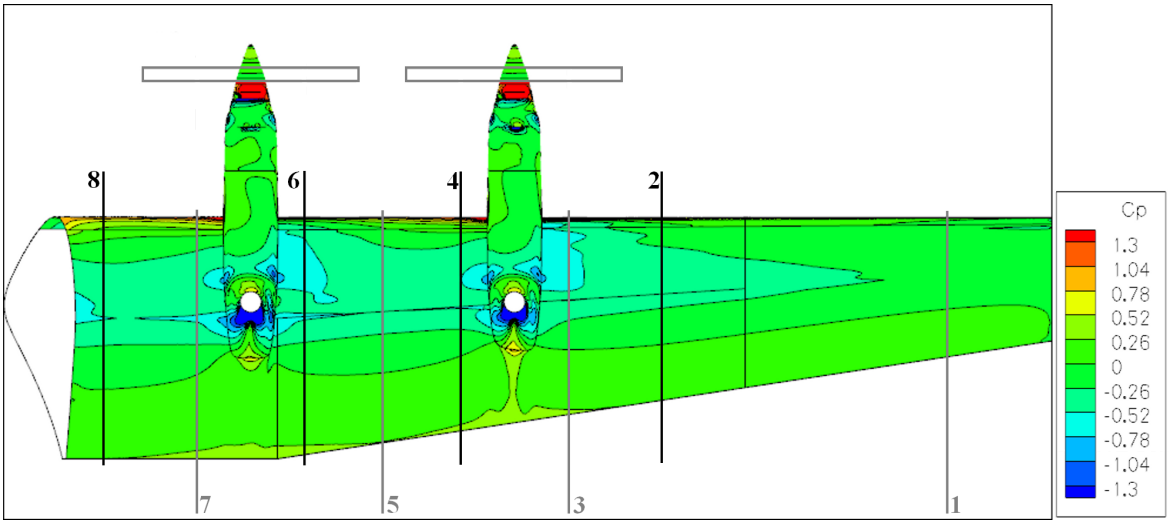


Figure 5.18: C_p plots and C_p isobars for Inboard-up rotating props case pressure side

identical to the prop off case. There is a very small decrease in static pressure for the suction side until 72% chord. Hence the outboard up rotation of the outboard engine seems to have a slight effect on section 1, although it is not located inside the propeller slipstream. This is in contrast with the inboard up case, where the C_p-plots for the prop off and inboard up case were identical. This also corresponds with Figure 5.11, which also predicted a larger C_l value for this section.

Considering section 2, there is a slight decrease in static pressure for the suction side and a slight increase in static pressure for the pressure side when compared to the prop off case. Although this de- and increase is very small, it does increase the sectional lift coefficient with respect to the prop off case.

At section 3 the influence of the outboard up rotating propeller becomes more apparent. Especially the suction side of the airfoil is affected, leading to a lower pressure on the upper side of the airfoil. Especially the LE suction peak is increased w.r.t. the prop off case. Furthermore the pressure increases on the airfoils' lower side till about 20% chord, where it starts to diminish slightly until 45% chord. However the decrease in static pressure for the suction side is higher than the decrease in static pressure for the pressure side, leading to a higher sectional lift coefficient.

That part of the airfoil that is located close to the LE of the wing produces negative lift for section 4. In contrast to section 3 the propeller slipstream seems to affect mainly the pressure side leading to more negative C_p-values until 65% chord where it starts becoming more positive. The upper surface is displaying a decrease in LE suction peak and a local C_p peak at 45% chord. This behaviour leads to a lower sectional lift coefficient and this is also confirmed by Figure 5.11

For the section located in between both engines, the C_p-values on the pressure side are more negative from 10 to 60% chord. There is also a slight increase in static pressure for the upper side until 10% chord. While the suction side has more negative C_p-values from 25 till 70% chord. Hence the sectional lift coefficient for the outboard up case is higher than the one of the prop off case.

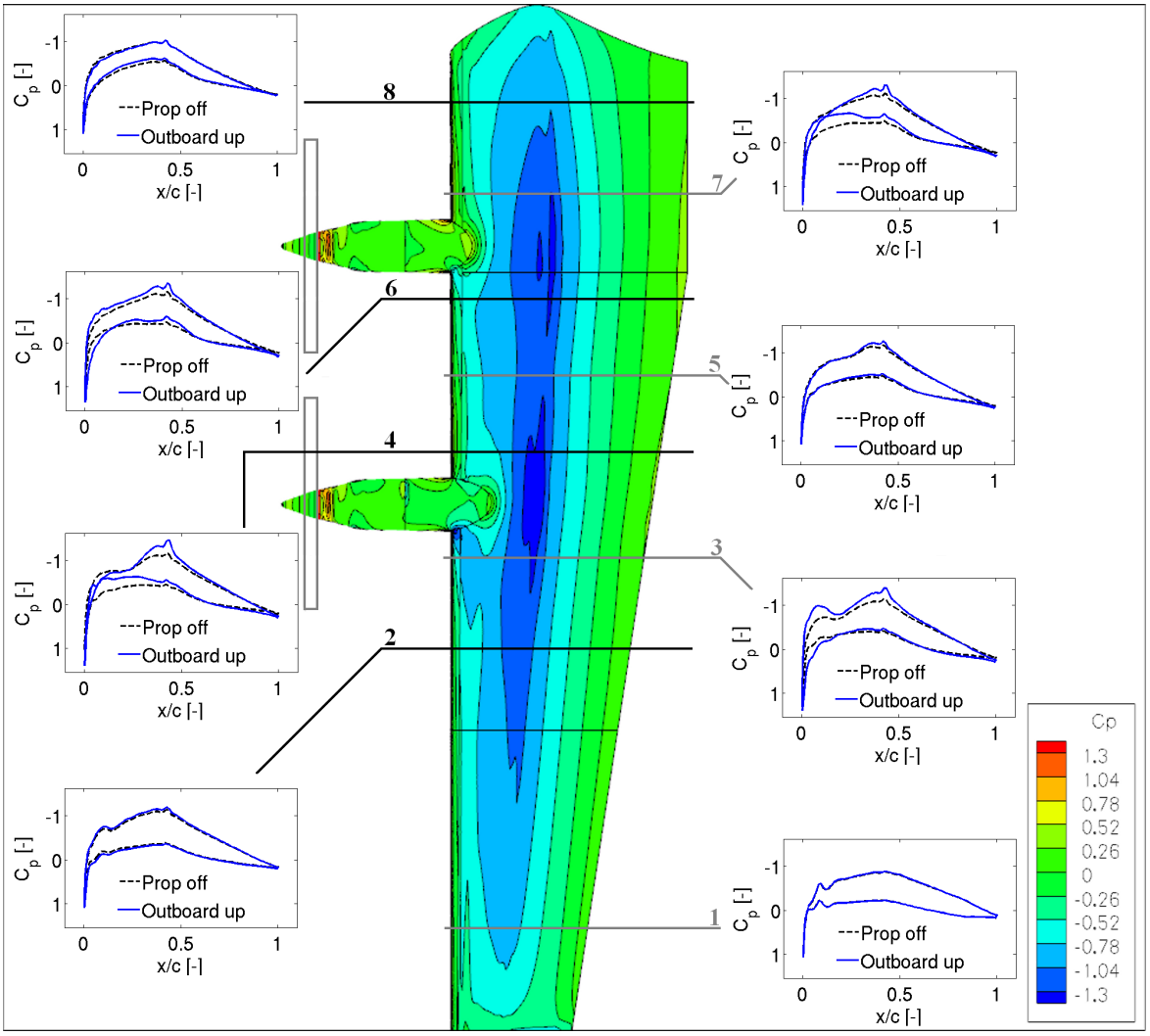


Figure 5.19: C_p plots and C_p isobars for Outboard-up rotating props

Outboard of the inboard engine, the pressure side displays the same behaviour as the lower side of section 3. The suction side produces more negative pressure coefficients until 75% chord, while there is a strong increase in static pressure for the suction side close to the LE. This translates into a higher sectional lift coefficient. However this sectional lift coefficient is lower than the one observed for section 3.

The changes in C_p for section 7 of the outboard up case with respect to the prop off case is similar as the changes that were noticed for section 4. Again a negative lift is produced close to the LE of the wing. The decrease in static pressure of the suction side is however less pronounced than observed for section 6. This leads to a strong decrease in sectional lift coefficient when compared to the prop off case and it is lower than the C_l of section 4.

Although section 8 is located outside the slipstream of the inboard propeller, its C_p-values are different than the prop off case. The C_p-values become more negative for the pressure side and more positive for the suction side, leading to a decreased sectional lift coefficient.

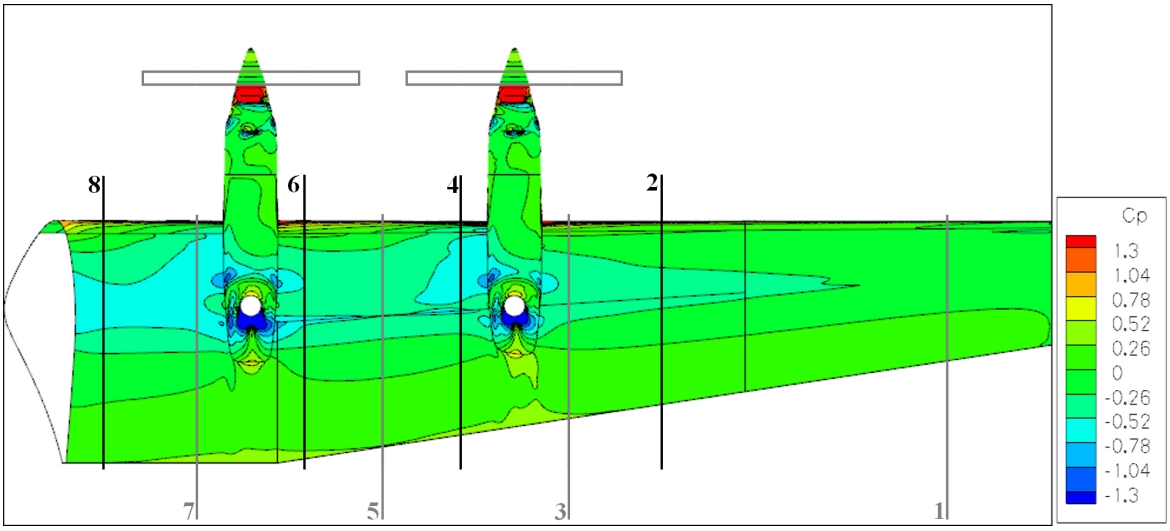


Figure 5.20: C_p plots and C_p isobars for Outboard-up rotating props case pressure side

In summary, sections 4, 7 and 8 produce a smaller lift coefficient when compared to the prop off case. Furthermore the static pressure distribution close to the TE of the wing changes more quickly to a higher value for sections located between section 4 and 7, which explains the lower value for the TE point in the pressure plots of sections 4 and 7. It also appears that the pressure side of the wing sections located behind the downward going blade is more affected by the propeller slipstream than the suction side. The opposite is true for he sections located in the upwash region of the propeller. The same observation was made for the inboard up case.

DBE rotation

By inspection of C_p-plot 1 from Figure 5.21 it can be seen that the distribution of the pressure coefficient for the airfoil of section 1 for the DBE rotation is almost identical to the prop off case. There is a very small decrease in static pressure for the suction side. Hence the outboard up rotation of the outboard engine seems to have a slight effect on section 1, although it is not located inside the propeller slipstream. This corresponds with the outboard up case, where a similar observation for the C_p-plot of section 1 was made.

The pressure coefficient plot of section 2 seems identical to section 2 of the outboard up case, however there is a very small difference. For the outboard up case, there is slightly more suction on the suction side and the static pressure is slightly larger for the pressure side. Hence the local lift coefficient of the DBE case is slightly less than the one of the outboard up case but it is larger than the sectional lift coefficient of the prop off case.

The pressure coefficient plot for the section 3 seems identical to the pressure plot of the outboard up case, however there is a small difference. The C_p values for the upper section is less negative for the DBE case. Hence the local lift coefficient is slightly lower than the one of the outboard up case but larger than C_l of the prop off case.

Although the C_p-plot from section 4 of the DBE case looks similar to the outboard up case, they are not identical. The main difference is the slightly larger suction peak at 45% chord and the slightly more negative C_p for the pressure side concerning the DBE case.

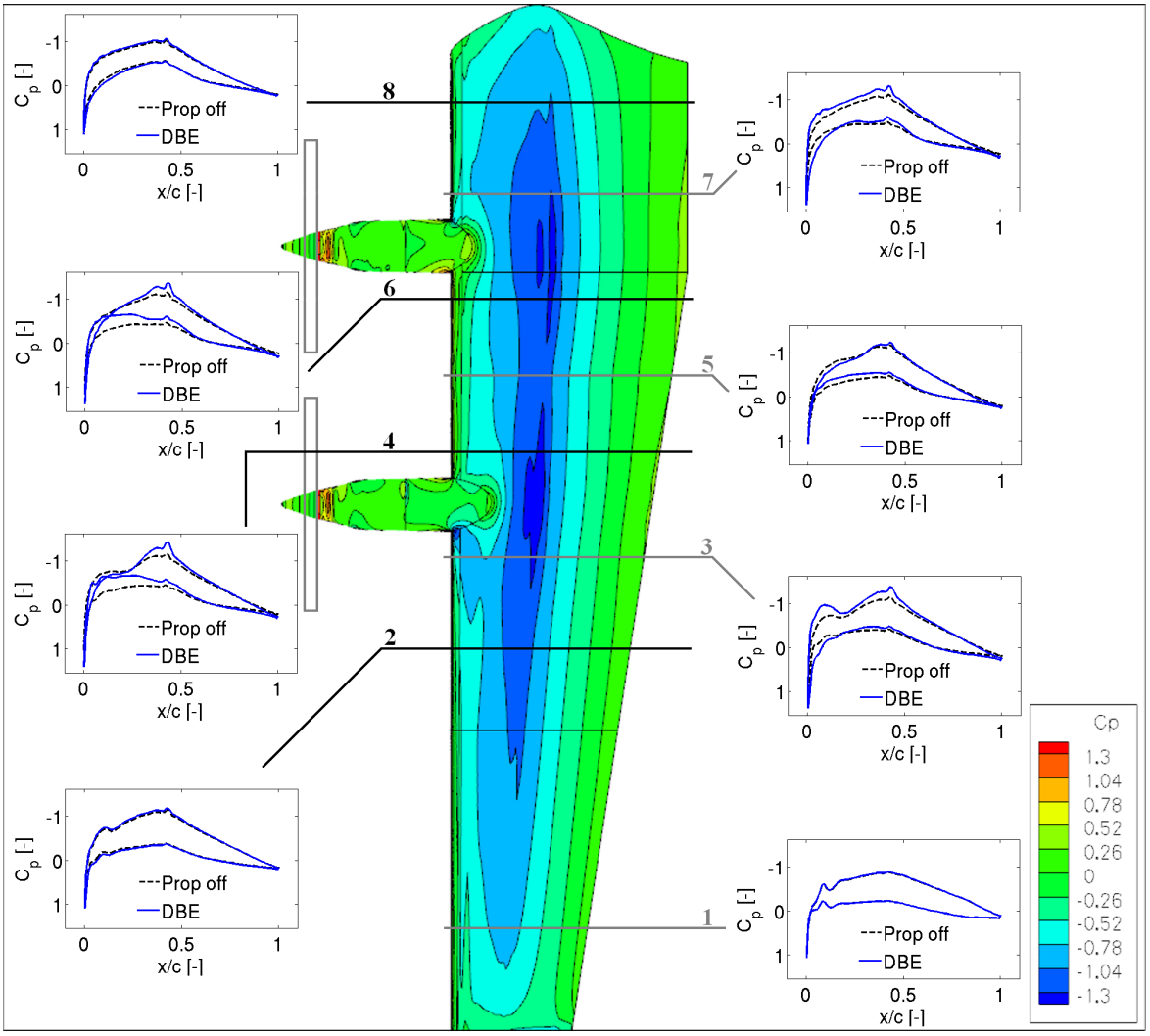


Figure 5.21: C_p plots and C_p isobars for DBE rotating props

This leads to a lower sectional lift coefficient than the outboard up case and hence it is also lower than the one of the prop off case.

The section located in between both engines displays neither the behaviour of the outboard up nor the inboard up case. The static pressure at the suction side is increased until 25% chord where it starts to decrease slightly until 65% chord. While the static pressure is diminished on the lower surface until 70% chord. The overall sectional lift coefficient is less than the prop off case and hence it is also lower than the inboard and outboard up case. This is also confirmed by Figure 5.11

With respect to the inboard up rotating case, section 6 of the DBE simulation has a slightly larger static pressure for the suction side and a smaller static pressure for the pressure side. Therefore for this airfoil section the DBE case has a lower sectional lift coefficient than the inboard up case and the prop off case.

The C_p-plot of section 7 of the DBE simulation has the same shape as the C_p-plot of the inboard up rotating case. However the increase in suction of the upper side with respect

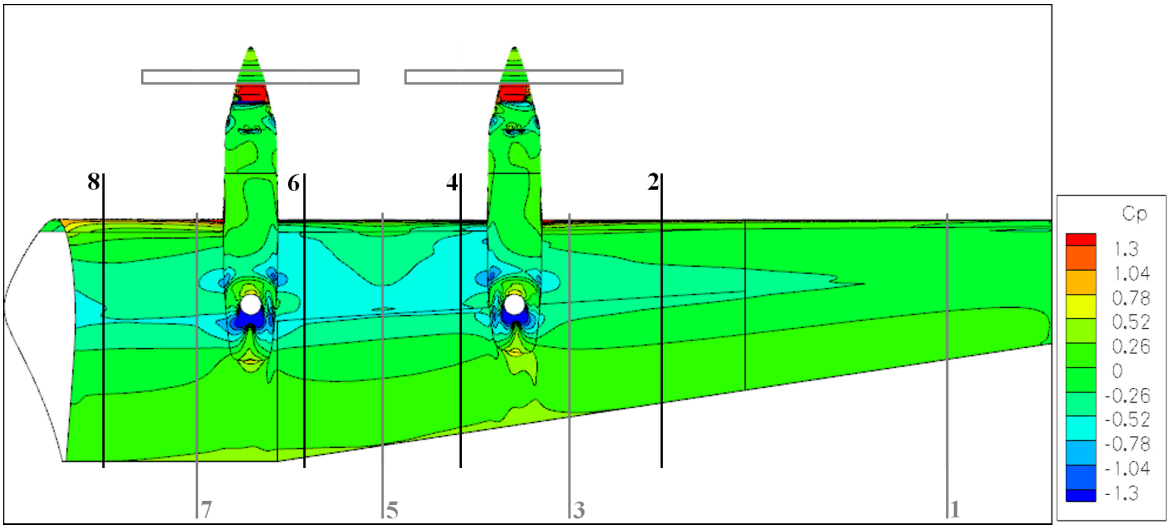


Figure 5.22: C_p plots and C_p isobars for DBE rotating props case pressure side

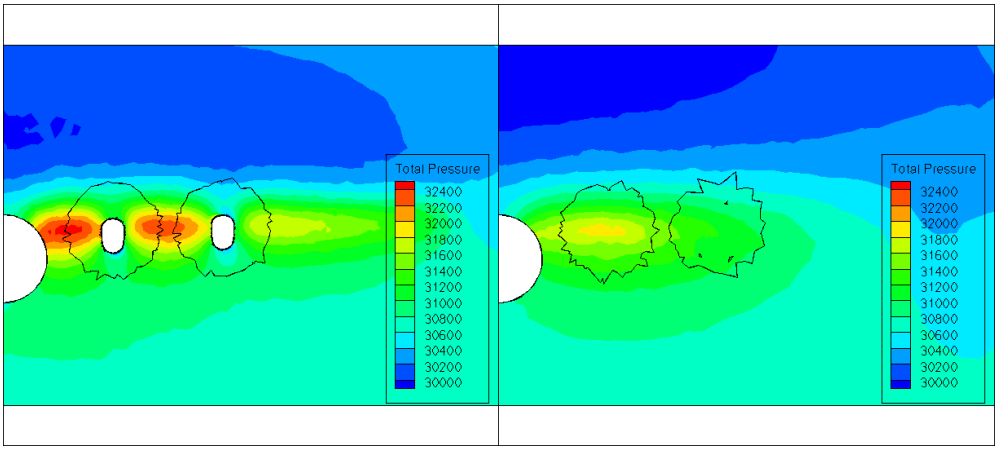
to the prop off case is less and the C_p-values at the pressure side are less positive. Hence this section will produce a lower lift coefficient compared to the inboard up case but it is still larger than the prop off case.

For section 8 one can again state that the shape of the pressure coefficient distribution is identical as the inboard up case, however the increase in suction on the suction side is less. And the C_p values on the pressure side seem to be more negative for the DBE case with respect to the inboard up case. This translates into a lower sectional lift coefficient for the DBE case with respect to the inboard up simulation, however it will be larger than the prop off case.

In summary the DBE case displays a similar behaviour as the outboard up configuration for sections 1 to 4. However the sectional lift coefficient for these sections are lower than the ones produced by the outboard up case. But they are higher than the C_l values of the prop off case for sections 1 to 3. The C_p-plots of the DBE case follow the same trend as the inboard up case for sections 6 to 8. The sectional lift coefficients of these sections produced by the DBE case are however lower than the ones produced by the inboard up configuration, but higher than the prop off case for sections 7 and 8. At section 5, which is located in between both engines, the C_p-plot differs from the C_p-plot of the inboard up and outboard up case. At this section a lower local lift coefficient is produced w.r.t. the prop off case but also w.r.t the inboard and outboard up case. All these observations correspond with the ones observed for the spanwise lift coefficient (Figure 5.11).

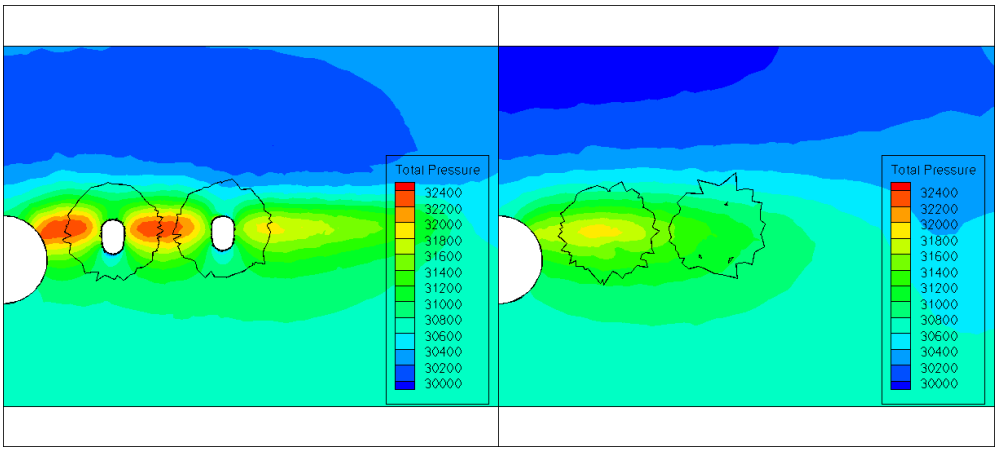
5.5.3 Influence of the wing on the total pressure

The actuator introduces a sudden jump in total pressure as could also be observed from Figure 4.13 of section 4.4.5. This corresponds to a more energetic flow due to the work delivered by the propeller. The influence of the wing on the total pressure can be deduced by inspection of the pressure contours in a plane 1 m upstream and downstream of the wing. The change in total pressure contours for the inboard up, outboard up and DBE case are given in respectively Figure 5.23 , 5.24 and 5.25.



(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing

Figure 5.23: Total pressure contours for the inboard up case.



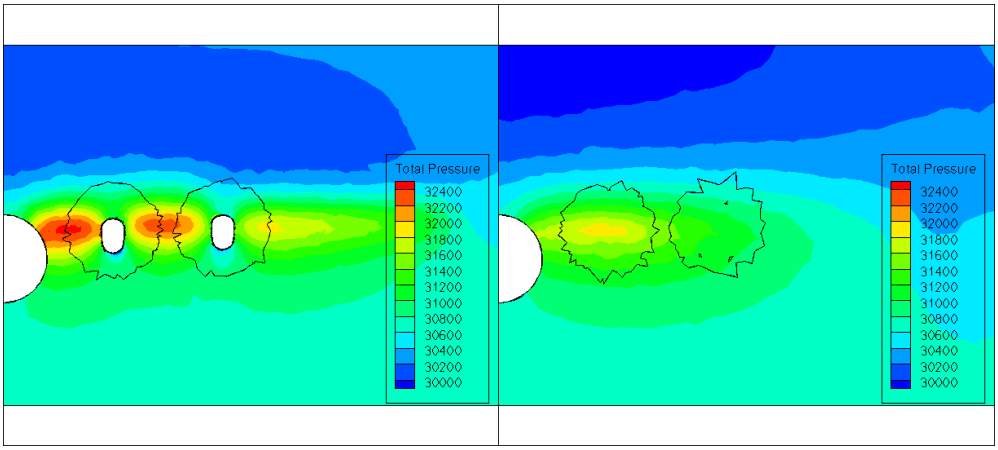
(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing

Figure 5.24: Total pressure contours for the outboard up case.

By inspection of Figures 5.23 , 5.24 and 5.25 it can be clearly seen that the wing extracts most of the energy induced by the propeller slipstream as indicated by the decrease in total pressure for the plane located 1 m downstream of the TE of the wing. The positive interaction between the upward going blade and the fuselage causes high values in total pressure around the upper side of the fuselage as can be seen from Figure 5.23a. The total pressure contours upstream of the wing of the DBE case inboard of the inboard engine are similar to the contours of the inboard up case. For the outboard engine the DBE case displays a similar behaviour as the outboard up case.

5.5.4 Influence of the wing on the tangential velocity

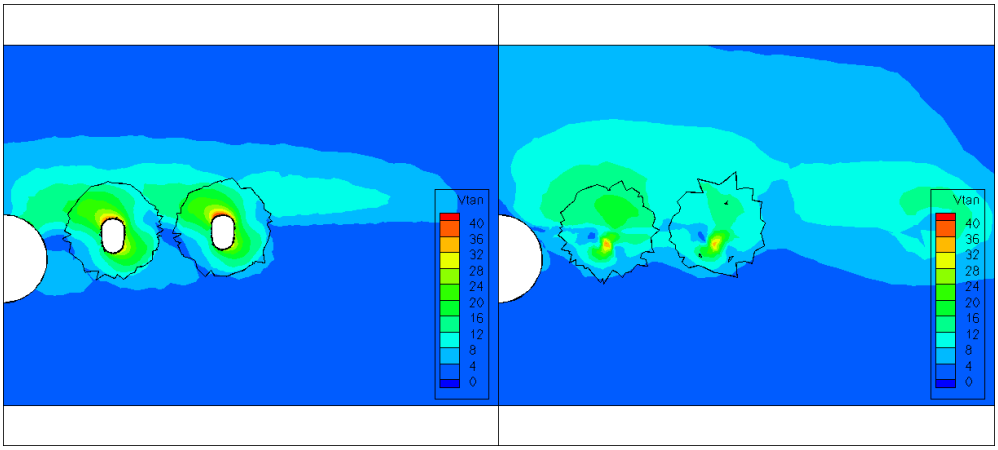
Upstream of the actuator disk the tangential velocity is zero, but at the actuator disk itself there is a jump in tangential velocity. Again planes are extracted 1 m upstream of the LE



(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing view

Figure 5.25: Total pressure contours for the DBE case.

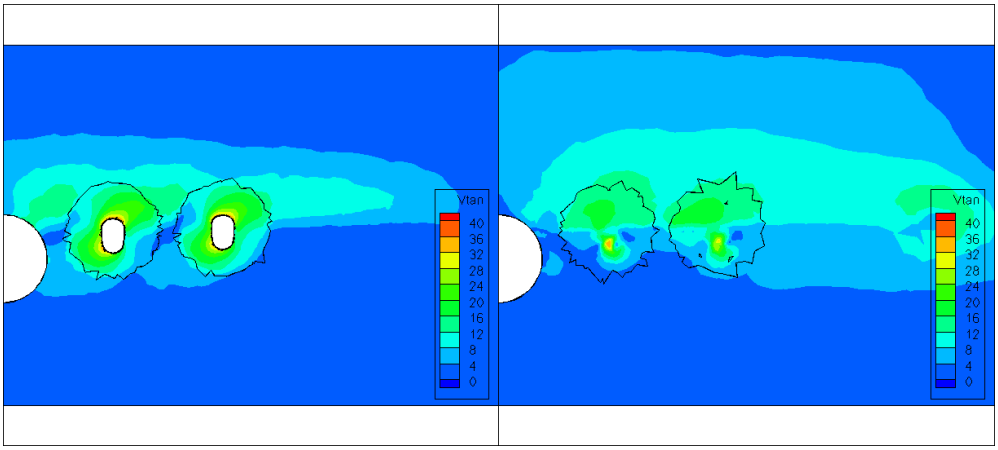
and 1 m downstream of the TE of the wing. The resulting tangential velocity distribution is given in Figures 5.26 , 5.27 and 5.28 for respectively the inboard up, outboard up and DBE case. It should be noted that these figures display the absolute value of the tangential velocity.



(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing

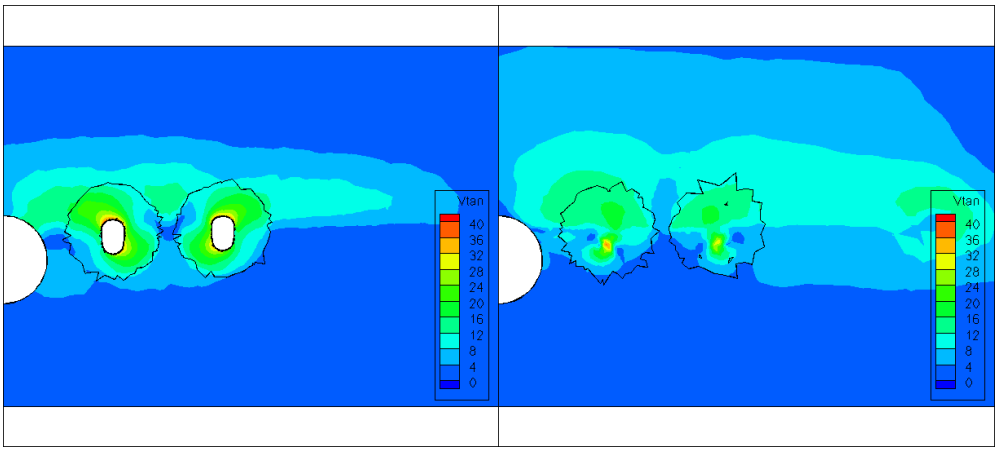
Figure 5.26: Tangential Velocity contours for the inboard up case.

From these figures it can be seen that the tangential velocity 1 m downstream of the TE of the wing is less than 1 m upstream of the wing's LE. Hence the wing acts as a stator. Furthermore it can be clearly seen that the tangential velocity contours around the inboard propeller of the DBE case resembles the ones around the inboard propeller of the inboard up configuration. On the other hand, the velocity contours around the outboard propeller of the DBE case is similar to the ones observed for the outboard propeller of the outboard up configuration. The tangential velocity distribution is not perfectly circular, but the upper and lower halves of the slipstream are shifted in opposite direction. This



(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing

Figure 5.27: Tangential Velocity contours for the outboard up case.



(a) 1 m upstream of LE wing (b) 1 m downstream of TE wing view

Figure 5.28: Tangential velocity contours for the DBE case.

phenomenon was also observed by Veldhuis [1996]. It should be noted however that the axes system of Figure 5.26b and reference Veldhuis [1996] differ, however corresponding results were obtained.

5.5.5 Comparison of airfoil data with XFoil

In order to verify whether the propeller-wing interaction study gives accurate results concerning the wing parameters C_l and C_d the software "XFOIL" was used on the root airfoil section. The imported file in XFOIL contains the x- and z-coordinates of the root profile. A viscid calculation was performed with the same Mach and corrected Reynolds number (Drela & Youngren [2001]) to match the same flow conditions of the propeller-wing interaction study. The user can impose a certain local lift coefficient in XFOIL in order to find the corresponding local drag coefficient and pressure coefficient distribution.

The airfoil wing section of the FLUENT model that was used as comparison is located at $y/y_{ref} = 0.124$ and its position is indicated in Figure 5.29. The lift coefficient for this airfoil was extracted from Figure 5.11 and the drag coefficient is the parasite drag coefficient as was obtained from the spanwise integration of pressure and friction forces for the prop off case since the propeller and 3D effects can not be included in XFOIL.
Since 3D effects are not included by the XFOIL calculation, also a 2D FLUENT calculation on the same airfoil section was performed.

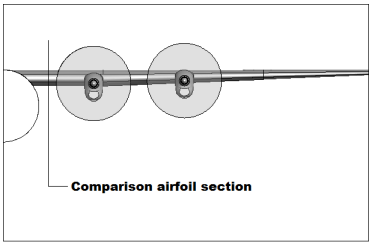


Figure 5.29: Position of "comparison" root section in Fluent.

The C_p-distribution calculated by XFOIL and the one obtained by a 2D and 3D simulation in FLUENT is displayed in Figure 5.30.

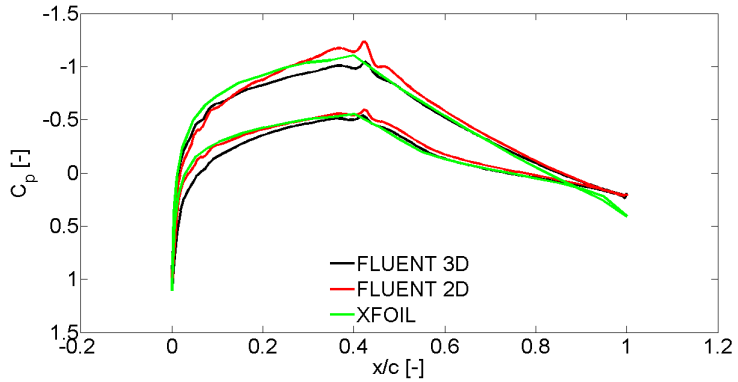


Figure 5.30: C_p distribution for the wing's root section as obtained By Fluent 3D, 2D and XFOIL.

As can be seen from Figure 5.30 the C_p-distributions are similar, however they are not identical. As expected, the XFOIL results correspond the best to the 2D FLUENT C_p-distribution. From Figure 5.29 it can be seen that the airfoil section is located in between the fuselage and the propeller tip. Hence a possible explanation for the discrepancy in C_p-plots is the fact that the upwash of the fuselage is not incorporated in the XFOIL and 2D FLUENT calculation.
The comparison of the sectional drag coefficient for the prescribed lift coefficient is given in Table 5.10. In this table also a difference is made between "matlab" and "fluent" 2D FLUENT results. By which is implied that the "matlab" results are obtained by chordwise integration of pressure and friction forces while the "fluent results" are obtained by making use of "force report" in FLUENT. This is done to assess the performance of the created m-file (Appendix C).
From Table 5.10 it can be concluded that all calculation methods predict different values

for the local drag coefficient of the wing's root. Even the 2D FLUENT(fluent) and XFOIL calculation produce different results. This is partially caused by the slightly higher lift coefficient of the 2D FLUENT(fluent) simulation. However the density and quality of the mesh also plays an important factor in the drag calculation of FLUENT. Furthermore the domain of the 2D FLUENT calculation was larger than the domain of the 3D FLUENT calculation. This implies that the calculation of the drag coefficient can be influenced by the distance of the airfoil w.r.t. the boundaries of the calculation domain. However this does not compromise the general conclusions that will be drawn from the inboard up, outboard up and DBE simulations since they are all performed on the same mesh. So the "error" introduced by a boundary proximity is probably similar for all cases. But it should be noted that a 2-dimensional approximate of 3-dimensional flow should always be considered with care. Furthermore the m-file which predicts the local lift and drag coefficient by means of chordwise integration of pressure and friction forces produces good results.

Table 5.10: sectional drag coefficient of the root section: XFOIL vs 2D & 3D FLUENT.

Case	$\mathbf{C_L}$ [-]	$\mathbf{C_D}$ [-]
FLUENT(matlab) 3D	0.3427	0.0175
FLUENT(matlab) 2D	0.3418	0.0063
FLUENT(fluent) 2D	0.3440	0.0067
XFOIL	0.3427	0.00409

5.5.6 Conclusions

This section will summarize the main observations and conclusions that were drawn in sections 5.5.1 and 5.5.2.
The inboard up case has the best aerodynamic performance since it has the highest lift-to-drag ratio and will therefore be able to cover a larger range than the other configurations. Furthermore its endurance is also the longest since the term $C_L^{3/2}/C_D$ is the highest of all cases considered. The main reason for these better performance parameters is the higher lift production while still maintaining a reasonable drag coefficient. This can be clearly seen by inspection of Table 5.8, which summarizes the percent in- or decrease in overall lift and drag coefficient with respect to the prop off case as obtained by using "force report" in FLUENT. From this table it can be seen that the slipstream generated by two adjacent inboard up rotating propellers is responsible for a 5% increase in overall lift coefficient and an augmentation of 30.8% of the overall drag coeffcient. Although the DBE case has the lowest overall drag coefficient the increase in overall lift coefficient w.r.t. the prop off case is only 1.5 %. This leads to a worse lift-to-drag ratio and endurance term than the inboard up case, but it will have a better aerodynamic performance than the outboard up case.
By inspection of the pressure coefficient plots of several sections along the wing it was discovered that for the wing sections located behind the upward going blade mainly the upper (suction) side of the wing is affected by locally decreasing the static pressure. Hence

producing a more negative pressure coefficient for the suction side, leading to a higher lift coefficient. The opposite is true for the wing sections located behind the downward going blade, where mainly the lower (pressure) side of the wing is affected by the propeller slipstream. This leads to a lower sectional lift coefficient, since the static pressure on the pressure side is decreased. The same conclusions can be found in references Moens & Gardarein [2001], Colin et al. [1996], Barber & Nelson [1996], Zang et al. [2001] and this was also predicted by theory. The effects induced by the propeller are of course most noticeable inside the propeller slipstream, however they are not limited to the wing sections located within this slipstream. Due to the spanwise gradients of the wing, the wing sections located outside the slipstream are also affected. Whether this has a positive or negative effect on the local lift production depends on the fact whether the wing section is located close to an upward or downward going blade. Again the downward going blade causes a decrease in local lift coefficient while the upward going blade induces an augmentation of the sectional lift coefficient.
The DBE case appears to follow the same trend concerning the pressure distribution as the outboard up case from the wing's tip till the section is reached which is located in between both engines. Here the pressure coefficient plot displays a different behaviour than both the inboard as outboard up case. From here on until the root of the wing, the pressure distribution of the DBE case will follow the same trend as the one observed for the inboard up configuration. However the local lift coefficient will always be less than the one observed for the inboard up (root to middle engines) and outboard up (middle of engines to tip) case. Furthermore the spanwise lift coefficient distribution of the DBE configuration is similar as the one predicted in Malard et al. [2005].
The same behaviour was observed by considering the spanwise distribution of lift (Figure 5.11) and drag (Figure 5.12) coefficient which was obtained by a chordwise integration of the pressure and friction forces acting on the wing.

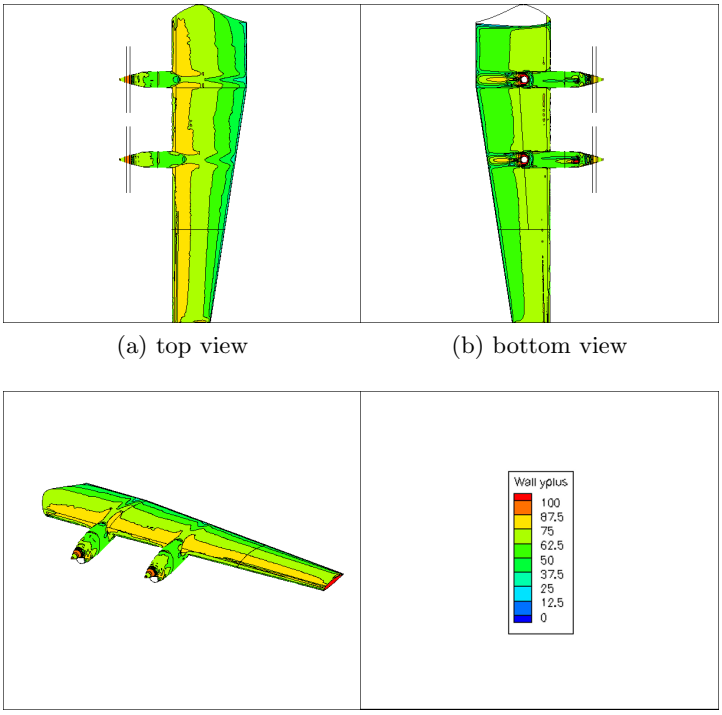
5.6 Quality check of the mesh

5.6.1 Boundary layer assessment

As discussed in section 5.1.2 the y^+ value is a parameter which states whether the boundary layer mesh is suitable to resolve turbulent flow. The height of the first cell is directly related to the y^+ parameter. For a first cell height of 1 mm, y^+ equals 80. To check whether the created boundary layer is indeed suitable to resolve turbulent flow, the y^+-values were determined for the prop off case and are depicted in Figure 5.31. Since all cases use the same mesh the results are similar for the inboard up, outboard up and DBE case. From Figure 5.31 it can be deduced that the created boundary layer is suitable to resolve turbulent flow since FLUENT's non-equilibrium wall function requires $y^+ \epsilon$ [30 300].

5.6.2 Grid independence check

In order to ensure that the results of the propeller-wing interaction study are valid in general, a grid dependency check has been performed. To this end a finer mesh was created. This mesh has the same surface mesh as the "original" mesh, however the



(a) top view (b) bottom view

(c) 3D view (d) legend

Figure 5.31: y^+ contours of the aircraft model.

growth factor used to grow cells from the boundary layer to fill up the domain is 1.2 instead of 1.4. This leads to a total number of cells of 11 245 973. Furthermore the refinement of the cells situated in the cylinders behind the propellers is different: instead of using the procedure described in section 5.1.2, the refinement in the cylinders depends on the maximum volume change. This means that only those cells will be refined which violate the maximum volume change. For this mesh a maximum volume change of 1.2 was chosen, which led to a very fine mesh which contains 18 865 930 cells in total. A formal procedure to check grid independency is described by Celik [2008]. This method is also recommended by the "Journal of Fluids Engineering" in its editorial policy. In order to follow this procedure, a third grid is necessary, which is more coarse or finer than the other meshes. Since the maximum number of cells has already been reached by the grid which contains almost $19 \cdot 10^6$ cells because of the considered cost, the third mesh will be more coarse. Instead of creating a new mesh, the mesh without refinement in the cylinders behind the propellers will be used. Thus three different meshes are compared:

1. "Coarse mesh" (Contains about 6 million cells)
2. "Original mesh" (The mesh used in the propeller-wing interaction study. Contains almost 11 million cells)
3. "Fine mesh" (Contains almost 19 million cells)

For the comparison of the three different grids only the prop off case is considered in order to limit the calculation cost (time and recources). Although the prop on cases induce a

different flow field, the trends that are visible for the different grids for the prop off case are expected to be similar for the prop on cases.
Besides the formal procedure, another way to check whether the results are mesh independent is to compare important parameters for all three grids and see how they differ. Since the lift and drag coefficients are the most important parameters in this study the overall lift and drag coefficients (Table 5.11) of all three different meshes will be compared as well as their spanwise distribution (Figure 5.32 and 5.33).

Table 5.11: Lift and drag parameters of the wing as obtained by Fluent

Case	L [N]	D [N]	C_L [-]	C_D [-]	ΔC_L [-]	ΔC_D [-]
Fine mesh	186593	12369	0.4125	0.0273	-0.2%	-0.6%
Original mesh	186974	13155	0.4134	0.0291	-	-
Coarse mesh	176003	22631	0.3891	0.0500	-6%	+42%

From Table 5.11 it follows that the difference in overall lift coefficient is negligible for the original and fine mesh while the coarse mesh gives an underestimation of 6%. Furthermore it can be noticed that the overall drag coefficient is more influenced by grid changes. The original mesh overestimates the overall drag coefficient with 6% w.r.t. the fine mesh, while the difference with the coarse mesh is 42%.
The spanwise lift and drag coefficients can be found respectively in Figures 5.32 and 5.33. Although the overall lift coefficient is almost similar for the original and fine mesh, locally there are some differences as can be seen from Figure 5.32. For the original and fine mesh the main differences occur close to the nacelle, while smaller differences can be observed for the wing regions situated inside the propeller diameter. Outside these regions the local C_L-values are identical. Apparently the difference in local lift coefficient is detected more accurately in the refinement region of the cylinders behind the propellers, although the size of the cells located outside these cylinders differ as well.

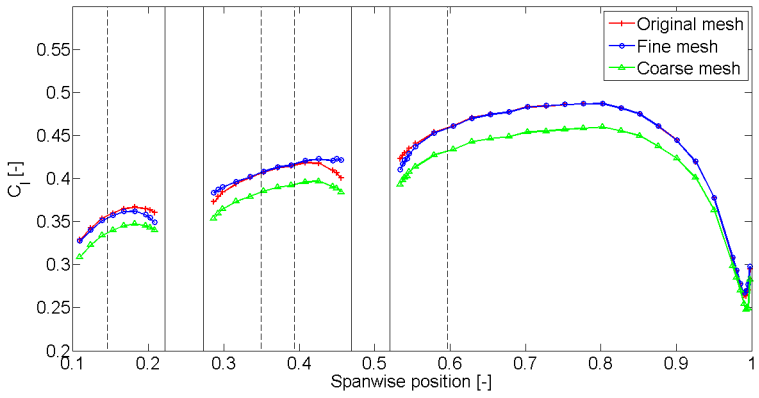


Figure 5.32: Spanwise lift coefficient ("—" nacelle borders, "- -" propeller diameter)

As can be seen from Figure 5.33, the original mesh values predicts local drag coefficients which are larger than the ones predicted by the fine mesh and smaller than the ones obtained by the coarse mesh. The overall trend of the original mesh agrees well with the

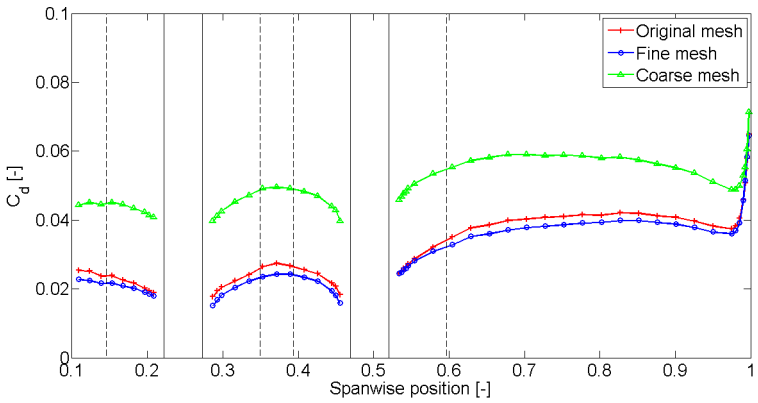


Figure 5.33: Spanwise drag coefficient ("—" nacelle borders, "- -" propeller diameter)

fine mesh even in the regions situated in the cylinder refinement zones. This is in contrast to the local lift coefficients and rather unexpected since from the overall drag coefficient values it appeared that the drag coefficient is more susceptible to grid changes.

For the formal procedure to check grid dependency the following parameters are important:

- The approximate relative error, e_a
- The extrapolated relative error, e_{ext}
- The fine grid convergence index, GCI_{fine}

The approximate relative error can be computed with Eq (5.18):

$$e_a^{21} = \left| \frac{\phi_1 - \phi_2}{\phi_1} \right| \tag{5.18}$$

where ϕ indicates the local or global variable under consideration. The subscript indicates the grid, where 1 indicates the finest mesh and 3 the grid which is the coarsest one. Thus the term e_a^{21} is the approximate relative error when considering the finest(1) and original(2) mesh.

Formula (5.19) determines the extrapolated relative error, while the fine grid convergence index can be calculated with Eq (5.20)

$$e_{ext}^{21} = \left| \frac{\phi_{ext}^{21} - \phi_1}{\phi_{ext}^{21}} \right| \tag{5.19}$$

$$GCI_{fine}^{21} = \frac{1.25\ e_a^{21}}{r_{21}^p - 1} \tag{5.20}$$

where the term $r_{21} = h_2/h_1$. The parameter h is a representive cell, mesh or grid size. In case of a local variable the local cell size can be used, while for a global variable an

" average global" cell size is used. For this calculation the "Volume-weighted-average" cell volume was used of the fluid zone that lies outside the boundary layer and does not include actuator related volumes nor transition cells. Hence this zone consists only of thetrahedrons.
The term ϕ_{ext}^{21} represents the extrapolated value of grid 1 and 2 such that:

$$\phi_{ext}^{21} = \frac{r_{21}^{p}\phi_1 - \phi_2}{r_{21}^{p} - 1} \tag{5.21}$$

The power term p can be obtained by solving the system of equations (5.22) to (5.24) by using a fixed-point iteration.

$$p = \frac{1}{ln(r_{21})}\left|ln\left|\frac{\epsilon_{32}}{\epsilon_{21}}\right| + q(p)\right| \tag{5.22}$$

$$q(p) = \left(\frac{r_{21}^{p} - s}{r_{32}^{p} - s}\right) \tag{5.23}$$

$$s = 1 \cdot sign\left(\frac{\epsilon_{32}}{\epsilon_{21}}\right) \tag{5.24}$$

where the terms ϵ_{21} and ϵ_{32} can be determined as follows: $\epsilon_{32} = \phi_3 - \phi_2$ & $\epsilon_{21} = \phi_2 - \phi_1$

Above equations are applied to the relation between the first and second grid, for the relation concerning the second and third grid e.g. e_a^{32} a similar procedure as previously described needs to be followed. The results of this formal grid dependency check for the global variables C_L and C_D can be found in Table 5.12.

From Table 5.12 it follows that for grid 2 and 3 the numerical error in fine-grid solution is 0.3% and 7.5 % for respectively the overall lift and drag coefficient. This shows that the coarse mesh is not fine enough to correctly predict the overal drag coefficient. While the numerical uncertainty in the fine-grid solution for the overall lift and drag coefficient concerning grid 1 and 2 is 0%, which indicates that the "original mesh" can be considered as grid-independent. This could also be deduced from Table 5.11.

Table 5.12: Calculation of the discretization error for global variables C_L and C_D

parameter	$\phi = \mathbf{C_L}$	$\phi = \mathbf{C_D}$
N_1, N_2, N_3	18 856930, 10 683 076, 6012070	18 856930, 10 683 076, 6012070
h_1, h_2, h_3	0.0742, 0.0870, 0.0872	0.0742, 0.0870, 0.0872
r_{21}	1.1712	1.1712
r_{32}	1.0026	1.0026
ϕ_1	0.4125	0.0273
ϕ_2	0.4134	0.0291
ϕ_3	0.3891	0.0500
p	$1.2475 \cdot 10^3$	973.5477
ϕ_{ext}^{21}	0.4125	0.0273
ϕ_{ext}^{32}	0.4143	0.0273
e_a^{21}	0.0021	0.0636
e_a^{32}	0.0587	0.7204
e_{ext}^{21}	0	0
e_{ext}^{32}	0.0023	0.0636
$\mathbf{GCI_{fine}^{21}}$	**0**	**0**
$\mathbf{GCI_{fine}^{32}}$	**0.0028**	**0.0747**

Conclusions and recommendations

A propeller-driven aircraft in tractor configuration produces a complicated flow field since there is a mutual interaction between the propeller and other aircraft parts. The propeller induces an increase in axial velocity on top of a tangential and radial velocity. If the airplane geometry is designed carefully a decrease in wing's drag can be obtained. The upwash generated by the wing alters the inflow angle of the propeller blade and hence the propeller thrust and torque. On the other hand the increase in axial velocity inside the propeller slipstream increases the drag production of the nacelles. Furthermore the nacelles introduce a blockage effect, which translates into a radial varying inflow velocity at the propeller and hence the thrust and torque production of the propeller is altered.
From the validation simulations performed in chapter 4 it followed that the uniform and non-uniform actuator disk model were the best numerical models to simulate propeller induced flow. Especially concerning the tangential velocity they outperformed Phillips' equations. Although the UDF involving Phillips' equations does not impose a radial velocity, the prediction of the radial velocity is surprisingly accurate since it is only slightly worse than the prediction made by both actuator disk models. Since the non-uniform actuator disk model is more complex and more expensive in terms of computational cost and resources the uniform actuator disk model was chosen as the numerical model to simulate propeller induced flow.
The results of the propeller-wing simulations showed that the configuration with two adjacent propellers which rotate inboard up has the best aerodynamic performance in terms of lift-to-drag ratio and endurance. The outboard up case produced the lowest overall lift coefficient and the highest overall drag coefficient, hence it has the worst aerodynamic performance of all cases. The Down-Between-Engine (DBE) configuration had a lower overall lift and drag coefficient than the inboard up case which led to performance parameters which are better than the outboard up case but worse than the inboard up configuration. However it is recommended to investigate the stability and control characteristics of the airplane before making a choice concerning rotation sense to verify whether the better performance of the inboard up case is indeed more valuable than the better stability characteristics of the DBE configuration in case of engine failure.
From the spanwise distribution of lift and drag coefficient and several pressure coefficient plots of different wing sections it could be concluded that the DBE configuration follows

the same trend as the inboard up case from root till in between both engines, were it deviates from the inboard and outboard up configuration. From here on till the wing tip it displays the same behaviour as the outboard up configuration. Wing sections which are located in the slipstream of a downward going blade experience a decrease in local lift coefficient. The opposite is true for the sections situated behind an upward going blade: their sectional lift coefficients are increased. The same conclusions can be found in references Moens & Gardarein [2001], Colin et al. [1996], Barber & Nelson [1996], Zang et al. [2001] and this was also predicted by theory. Furthermore the spanwise lift coefficient distribution of the DBE configuration is similar as the one predicted in Malard et al. [2005]. It should be noticed however that despites the fact that the inboard case delivers the better overall lift coefficient, the outboard up and DBE configuration have a better distribution of the local lift coefficient towards the tip, since the local C_l values are higher than the prop off and inboard up configuration.

The numerical model used to predict propeller-wing interaction effects is only valid for a zero angle of attack of the propeller. Due to the upwash produced by the wing this assumption is no longer valid. This is the major drawback of this model as the action of the wing on the propeller is not included. The non-zero propeller angle of attack can be included in the code by predicting the average change in inflow angle per quadrant of the actuator disk. However this is also a crude approximation of the real flow and it is recommended to investigate this matter more thorough. Furthermore it has to be investigated whether the calculations which are necessary to predict the inflow angle of the propeller can be incorporated in a UDF for FLUENT since the macro commands limit the user.

Although the propeller model assumes steady, incompressible and inviscid flow, these assumptions do not compromise the results of the simulations since they are only applied to the actuator volume while the rest of the domain was simulated by making use of RANS. However it is recommended to investigate whether the current propeller model can be further improved by incorporating unsteady effects.

Despite the zero propeller angle of attack and steady flow limitation, interesting flow features of the steady part of the flow field are retrieved and this explains the success of comparable methods in recent designs (Moens & Gardarein [2001]). The strong point of the UDF is the fact that it only needs a few parameters (e.g. propeller diameter and thrust) as input variables. Furthermore the computational cost is low while the results are quite accurate. Therefore the UDF is already applicable in the preliminary design stage of an aircraft.

Bibliography

ANSYS FLUENT 12.0 Theory Guide [Computer software manual]. (2009). ANSYS, Inc.

ANSYS FLUENT 12.0 UDF Manual [Computer software manual]. (2009). ANSYS, Inc.

ANSYS FLUENT 12.0 User's Guide [Computer software manual]. (2009). ANSYS, Inc.

Barber, D., & Nelson, T. (1996). Measurements Of The Propeller Slipstream Interaction With A Nacelle And Wing. *ICAS*, *Vol.96-4.10.4*, 2414–2424.

Borst, H. (1981, April 7-10). Propeller Performance and Design as Influenced by the Installation. In *Business Aircraft Meeting & Exposition.* Wichita, Kansas.

Carlton, J. (2007). Marine Propellers And Propulsion. Elsevier Ltd.

Celik, I. (2008, July). *Procedure for Estimation and Reporting of Discretization Error in CFD Applications.* West Virginia University, Mechanical and Aerospace Engineering Department.

Colin, P., Moreux, V., & Barillier, A. (1996). Numerical Study Of Aerodynamic High Speed Propeller Engine Integration On Transport Aircraft. *ICAS*, *Vol.96-4.10.1*, 2366–2380.

Conway, J. (1995). Analytical solutions for the actuator disk with variable radial distribution of load. *Fluid Mechanics*, *Vol.297*, 327–355.

Davidson, P. (2004). *Turbulence: An introduction for scientists and engineers.* Oxford University press.

De Lathouder, A. (1948). *Luchtschroeven I.* Deventer.

Drela, M., & Youngren, H. (2001). XFOIL 6.9 User Primer [Computer software manual].

Durand, W., & Glauert, H. (1935). Airplane propelers. In *Aerodynamic Theory Vol IV.* Berlin: Springer.

Feng, J. (2001). Computational Analysis for an Advanced Propeller Powered Theater Transport. *AIAA*, *Vol.2003-4081*.

GAMBIT 2.4 Modeling Guide [Computer software manual]. (2007). Fluent, Incorporated.

Jansen, M. (1991, Mar.). *An evaluation of methods for the calculation of the flow around a propeller at angle of attack* (No. A-225). Fokker Aircraft B.V. Amsterdam.

Lino, M. (2010, March). *Literature study: Numerical investigation of propeller-wing interaction effects for a large military transport aircraft.* Technische Universiteit Delft.

Malard, L., J.M., B., Atinault, O., Larcher, J., & Sechaud, J. (2005, June 6-10). High Speed Testing and CFD Investigations For the New Generation Military Transport Aircraft Development. In *35th AIAA Fluid Dynamics Conference and Exhibit.* Ontario, Canada.

Marinus, B. (2007, Jun.). *Multidisciplinary evaluation of known propeller configurations* (No. 2007-15). Von Karman Institute.

Marinus, B. (2009, Aug). *Summary of a selected part of the lecture series "numerical investigation in turbomachinery: The state of the art" eddy viscosity models for turbulence and transition.* Numerical Investigation in Turbomachinery: The State of the Art von Karman Institute for Fluid Dynamics, Lecture Series 2009-08.

Miranda, L., & Brennan, J. (1986). Aerodynamic Effects Of Wing-Mounted Propellers And Turbines. *AIAA*, *Vol.86-37801*.

Moens, F., & Gardarein, P. (2001, June 11-14). Numerical Simulation Of The Propeller/Wing Interactions For Transport Aircraft. In *19th Applied Aerodynamics Conference.* Anaheim, CA..

Mullier, T. (2009). *Simulations numériques d' écoulement sur des parties d'aéronefs.* Msc Thesis, Royal Military Academy, Brussels, Belgium.

Phillips, W. (2004). Mechanics Of Flight. In (chap. 2.4: Propeller Momentum Theory). Hoboken, New Jersey: John Wiley & Sons, Inc.

Ruijgrok, G. (1996). Elements Of Airplane Performance. In (chap. 7: Propeller Performance). Delft University Press.

Sanchez-Caja, A. (2009). *Propeller Theory.* Lecture notes Helsinki University of Technology.

Veldhuis, L. (1996). Analysis of propeller slipstream effects on a trailing wing. *ICAS*, *Vol.96-4.10.3*, 2392–2413.

Veldhuis, L. (2005). *Propeller Wing Aerodynamic Interference.* Ph.D. Thesis, Technische Universiteit Delft, Delft, The Netherlands.

White, F. (2006). *Viscous Fluid Flow (Third Edition).* McGraw-Hill.

Witkowsky, D., Johnston, R., & Sullivan, J. (1989). Propellers/Wing Interaction. *AIAA*, *Vol.89-05325*.

Witkowsky, D., Lee, A., & Sullivan, J. (1988, January 11-14). Aerodynamic Interaction Between for Propellers and Wings. In *AIAA 26th Aerospace Sciences Meeting.* Reno, Nevada.

Zang, F., Khalid, M., J., Syms, & Ball, N. (2001, June 11-14). Numerical Investigation of Propeller Effects on the Aurora Aircraft. In *19th Applied Aerodynamics Conference.* Anaheim, CA..

Appendix A

SR-1 spinner-nacelle coordinates

Table A.1: Spinner-nacelle coordinates for the SR-1 propeller

x-coordinate	y-coordinate	z-coordinate
0.0	0	0.0
0.005	0	0.008
0.01	0	0.013
0.015	0	0.017
0.02	0	0.02
0.03	0	0.025
0.04	0	0.029
0.05	0	0.033
0.06	0	0.0365
0.08	0	0.043
0.1	0	0.05
0.125	0	0.0592
0.150	0	0.068
0.175	0	0.076
0.200	0	0.085
0.225	0	0.093
0.250	0	0.1
0.275	0	0.105
0.300	0	0.109
0.325	0	0.110
0.350	0	0.110
0.375	0	0.108

x-coordinate	y-coordinate	z-coordinate
0.400	0	0.105
0.425	0	0.103
0.450	0	0.101
0.475	0	0.1
0.500	0	0.0999
0.525	0	0.097
0.550	0	0.095
0.575	0	0.095
0.600	0	0.094
0.625	0	0.094
0.650	0	0.094
0.675	0	0.0945
0.700	0	0.095
0.725	0	0.096
0.750	0	0.096
0.775	0	0.096
0.800	0	0.096
0.825	0	0.096
0.850	0	0.096
1.000	0	0.096
1.5	0	0.096
5.0	0	0.096

Appendix B

UDF source codes

B.1 UDF uniform actuator disk

```
/*UDF for M=0.6 & J=3.08 @ 35000 ft, uniform Vin equal to freestream inboard engine*/

/*The following 2 commands are always necessary for the importation of a UDF in Fluent*/
#include "udf.h"
#include "stdio.h"

/*UDF description*/
/*This UDF will impose the x, y and z velocities at the actuator volume in order to simulate the effects of a rotating propeller by
using Conway's actuator disk model.
A UDF is employed for each engine. So the UDF needs to be adjusted accordingly. This can be done by adjusting the constant variables
inside the section "Define the constant variables" which differ for different cruise conditions and different geometry and mesh
set-ups. The variables related to the geometry are:  The (x,y,z) coordinate of the engine hub point, (xE,yE,zE)
The coordinates of the propeller leading (LE) and trailing (TE) edge (axial position and radius)

The variables related to the geometry are: The nr of cells in z-direction for the actuator volume, total_nr_cells_z
The zone_ID's assigned by Fluent to define the different zones of the mesh

The variables related to the cruise conditions are: The freestream velocity, VINF [m/s]
The angulear velocity of the propeller, omega [rps]
The total thrust of the propeller, T [N]
The density at the cruise altitude RHO [kg/m^3]
The rotation sense of the propeller, sense (1 indicates clockwise and 0 counter-clockwise)
The UDF is written by Marilyne Lino, Aerospace Engineering Msc student TU Delft and is last adjusted 25-02-10*/
/*End UDF description*/

/*****************************************************************************************************************************************
Define the constant variables
*****************************************************/
/*Variables that are geometry and mesh related*/
#define xE 1 /*x-coordinate of hub point of the engine wrt global coordinate system*/
#define yE 2 /*y-coordinate of hub point of the engine wrt global coordinate system*/
#define zE -1 /*z-coordinate of hub point of the engine wrt global coordinate system*/
#define RP 0.311 /*Actuator disk radius [m]*/
#define RLE 0.06 /*Distance from z=0 axis to nacelle contour at x=LE or radius of the nacelle at x=LE [m] (axisymmetric nacelle)*/
#define RTE 0.09 /*Distance from z=0 axis to nacelle contour at x=TE or radius of the nacelle at x=LE  [m] (axisymmetric nacelle)*/
#define total_nr_cells_z 42   /*Number of cells in radial direction (which is equal to z-direction(=along blade span) for y=yE)*/
#define nr_arc_cells 40   /* Number of cells in along the actuator's arc.*/
#define dx 0.006 /*width of the cells of the actuator volume in axial direction*/
#define outflow_zone_nr 28; /*ID_number for the zone of outflow of the actuator disk*/
#define flatdisk_zone_nr 14; /*ID_number for the zone of the flatdisk*/
/*End Variables that are geometry and mesh related*/

/*Variables that are related to cruise conditions*/
#define VINF 177.91 /*Freestream velocity [m/s].*/
#define omega 92.86 /*Propeller angular velocity (RPS)*/
#define T 197.23 /*Proppeller total thrust (N) for one blade was 15.068*/
#define RHO 0.3796     /*Air density at flight Altitude (kg/m^3)*/
#define sense 1        /*Sense=1 for clockwise rotating propellers and 0 for counter-clockwise rotating propellers*/
/*End Variables that are related to cruise conditions*/

#define PI 3.14159265358979323846 /*The number pi*/
/*****************************************************************************************************************************************/
/*****************************************************/
```

```
/************************************************************************************************************************************
Declarations of the global variables used in DEFINE_EXECUTE_AFTER_DATA and all the 3 DEFINE_PROFILE progs
************************************************************************************************************************************/
/*General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/
Domain *domain;
Thread *t;
face_t f;
cell_t c;
int zone_ID;   /*the term zone_ID is used to move the pointer from one computational zone to another*/
double x[ND_ND];  /*the vector x refers always to the coordinates of the centroid of a face or cell*/
/*End General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/

/*Counters in a loop*/
int i;
int j;
/*End Counters in a loop*/

/*Definition of the disk_inflow and disk_outflow thread pointers*/
Thread *disk_outflow; /*Used to move pointer to the outflow plane of the actuator disk*/
Thread *flatdisk; /*Used to move pointer to the flatdisk*/
/*End Definition of the disk_inflow and disk_outflow thread pointers*/

/*Variables needed to define the array sizes*/
int total_nr_faces_outflowface; /*total number of faces of the outflow plane of disk*/
/*End Variables needed to define the array sizes*/

double *v_A=NULL;
double *V_axial=NULL;
double *VT_z=NULL;
double *VR_z=NULL;
double *Vy=NULL;
double *Vz=NULL;

double OMEGA;   /*OMEGA =omega*2PI*/

double mean_v_A;
double sum_v_A;

/************************************************************************************************************************************
DEFINE_EXECUTE_AFTER_CASE UDF
************************************************************************************************************************************/
/*Use a DEFINE_EXECUTE_AFTER_CASE UDF to calculate the global variables total_nr_faces_inflowface and total_nr_faces_outflowface in
order to prevent mallocing, leading to less memory usage*/
DEFINE_EXECUTE_AFTER_CASE(Flatdisk_uniform_M06J308_validation, libudf_inb)
{

 #if !RP_HOST

 /**Calculation of the number of faces of the outflow plane of the actuator disk**/
 domain = Get_Domain(1);    /*The calculation domain needs to be specified at the beginning of the code*/
 zone_ID = outflow_zone_nr; /*ID_number for the zone of outflow of the actuator disk*/
 disk_outflow = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the outflow domain of the actuator disk*/

 /**Count nr of faces loop**/
 /**This loop runs over all the faces of the outflow plane, to count the total number of faces of the outflow plane of
 the actuator disk**/
 i=0;
 begin_f_loop_int(f,disk_outflow)
 {
  i=i+1;
 }
 end_f_loop_int(f,disk_outflow)

 total_nr_faces_outflowface=i;

 total_nr_faces_outflowface=PRF_GISUM1(total_nr_faces_outflowface);

 Message0("Content of variable total_nr_faces_outflowface is: %d\n", total_nr_faces_outflowface);
 /**End Count nr of faces loop**/
 /**End Calculation of the number of faces of the outflow plane of the actuator disk**/

 #endif /*!RP_HOST*/

}
/************************************************************************************************************************************
End DEFINE_EXECUTE_AFTER_CASE UDF
************************************************************************************************************************************/

/************************************************************************************************************************************
 VELOCITY IN X-DIRECTION
************************************************************************************************************************************/
DEFINE_PROFILE(x_vel_inboard, t1, indices)
{

 #if !RP_HOST

 /*****************************
 Declaration of local variables
 *****************************/
```

```
/*Mallocing global variables by ensuring that it is only malloced once hence the if statement*/
/*Variables which are also used globally needs to be malloced first.*/
if(!V_axial)
{
 Message0("Allocating V_axial\n");
 V_axial = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
}

 if(!v_A)
{
 Message0("Allocating v_A\n");
 v_A = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
}
 /*Mallocing global variables by ensuring that it is only malloced once.*/


/**Define variables belonging to the flatdisk**/
double v_i;
double Vas;
double omega_s;
double Rs;

double r_actuator[total_nr_faces_outflowface];
double vol;
double dy;
double dr_end[total_nr_faces_outflowface];
double drs[total_nr_faces_outflowface];
double rs[total_nr_faces_outflowface];
double difference_p[total_nr_faces_outflowface];
double dT_centroid[total_nr_faces_outflowface];

double check_thrust1;
double check_thrust;
double check_thrusttotal;
int divider;
int jtotal;

double vT_inf[total_nr_faces_outflowface];

double sum_v_Atotal;

/**End Define variables belonging to the flatdisk**/

/**Define all array elements related to total_nr_faces_outflowface equal to zero to ensure that computation works in parallel mode**/
for (j=0; j<=total_nr_faces_outflowface-1; j++)
{
 r_actuator[j]=0;
 dr_end[j]=0;
 drs[j]=0;
 rs[j]=0;
 difference_p[j]=0;
 dT_centroid[j]=0;
 vT_inf[j]=0;
}
/**End Define all array elements related to total_nr_faces_outflowface equal to zero to ensure that computation works in
parallel mode**/

/**Define files to write data for checking the UDF**/
/*FILE *file1;*/
/*FILE *file2;*/
/**End Define files to write data for checking the UDF**/

/*********************************
End Declaration of local variables
*********************************/


/**Calculation of the radii of the centroids inside the actuator volume and their corresponding elemental thrust**/
zone_ID = flatdisk_zone_nr; /*ID_number for the zone of the flatdisk*/
/*zone_ID = 14; /*ID_number for the zone of the flatdisk*/
flatdisk = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the flatdisk domain*/
OMEGA = 2*PI*omega;  /*Calculation of angular velocity in rad/s*/
/*Calculation of Phillips' parameters in order to calculate the elemental thrust of each elemental annular element*/
 v_i=sqrt((pow(VINF,2)/4)+((pow(OMEGA,2)*pow(RP,2))/4)*
(1-sqrt(1-((4*T)/(PI*(pow(RP,2)-pow(RTE,2))*RHO*pow(OMEGA,2)*pow(RP,2))))))-(VINF/2);
Vas = VINF+2*v_i;
omega_s = (4*(VINF+2*v_i)*v_i)/(OMEGA*pow(RP,2));
Rs =sqrt((VINF+v_i)/Vas)*RP;
/*End Calculation of Phillips' parameters in order to calculate the elemental thrust of each elemental annular element*/
/*file1 = fopen("check1.dat","w");*/
i=0;
begin_c_loop_int(c, flatdisk)
{
 C_CENTROID(x,c,flatdisk);   /*Calculates the cell's centroid*/
 r_actuator[i]=sqrt(pow((x[1]-yE),2)+pow((x[2]-zE),2));  /*Radius computation of centroid of the cells of the flatdisk*/
 vol=C_VOLUME(c,flatdisk);
 dy=(2*PI*r_actuator[i])/(8*nr_arc_cells);          /*The term 1/8 is included since only 1/8th of prop is modelled*/
 dr_end[i]=vol/(dx*dy);
 drs[i] =sqrt((VINF+v_i)/Vas)*dr_end[i];  /*Calculation of the elemental width drs*/
 rs[i] =sqrt(((VINF+v_i)/Vas)*(pow(r_actuator[i],2)));
```

```
  difference_p[i] = -(RHO*pow(omega_s,2)/2)*(pow(Rs,2)-pow(rs[i],2));
  dT_centroid[i] =(Vas-VINF)*RHO*Vas*((2*PI)*rs[i]*drs[i])+difference_p[i]*((2*PI)*rs[i]*drs[i]); /*Elemental thrust of an
  annular element*/

   /*fprintf(file1," %.6g \n",r_outflow_centroid[i]);*/
   i=i+1;
 }
  end_c_loop_int(c, flatdisk)
  /*fclose(file1);*/
 /**End Calculation of the radii of the centroids inside the actuator volume and their corresponding elemental thrust**/


   /**Check whether the thrust calculation is correct.**/
   /*The total thrust is divided over all the individual faces of the outflow plane.
   Since we have ttotal_nr_faces_outflowface different r_actuator, we will have total_nr_faces_outflowface different thrusts.
   The total thrust is equal to the sum of all the elemental thrusts of each face of the outflow plane.
   So the sum of all dT_centroid*total_nr_faces_outflowface must equal the total Thrust T*/
   check_thrust1 = dT_centroid[0];

   for (i=0; i<=total_nr_faces_outflowface-2; i++)
   {
    check_thrust1 = check_thrust1 + dT_centroid[i+1];
    check_thrust1 = check_thrust1;
   }

   divider=total_nr_faces_outflowface/total_nr_cells_z;
   check_thrust = check_thrust1/(total_nr_faces_outflowface/total_nr_cells_z);
   /*Check whether the thrust calculation is correct. All dT_centroids must be positive*/
   j=0;
   for (i=0; i<=total_nr_faces_outflowface-1; i++)
   {
    if (dT_centroid[i]>=0)
{
 j=j+1;
}
   }

  check_thrusttotal=PRF_GRSUM1(check_thrust);
  Message0("Content of divider is: %d\n", divider);
  Message0("Content of check_thrusttotal is: %f\n", check_thrusttotal);
  jtotal=PRF_GISUM1(j);
  Message0("Content of j is: %d\n", jtotal);
  /**End Check whether the thrust calculation is correct. The sum of all dT_centroid must equal T**/

 /**Calculation of the axial velocity at flatdisk**/
 /*file2= fopen("check2.dat","w");*/
 i=0;
 begin_c_loop_int(c, flatdisk)
 {
  vT_inf[i] = omega_s*rs[i];
  v_A[i] = 0.5*(-VINF + sqrt(pow(VINF,2) + pow(vT_inf[i],2) +dT_centroid[i]/(RHO*PI*r_actuator[i]*dr_end[i])));
  V_axial[i]= VINF +v_A[i];

  /*fprintf(file2,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \n", r_actuator[i], dT_centroid[i], vT_inf[i], v_A[i], V_axial[i]);*/
  i=i+1;
 }
 end_c_loop_int(c, flatdisk)
 /*fclose(file2);*/
 sum_v_A = v_A[0];

 for (i=0; i<=total_nr_faces_outflowface-2; i++)
 {
  sum_v_A = sum_v_A + v_A[i+1];
  sum_v_A = sum_v_A;
 }

 sum_v_Atotal=PRF_GRSUM1(sum_v_A);
 Message0("Content of sum_v_A is: %f\n", sum_v_Atotal);
 /**End Calculation of the axial velocity at flatdisk**/

 /**Impose the axial velocity using F_PROFILE**/
 i=0;
 begin_c_loop_int(c, t1)
 {
  F_PROFILE(c, t1, indices) = V_axial[i];
  i=i+1;
 }
 end_c_loop_int(c, t1)
 /**End Impose the axial velocity using F_PROFILE**/

  #endif /*!RP_HOST*/

}
/*****************************************************************************************************************************
End VELOCITY IN X-DIRECTION
*****************************************************************************************************************************/


/*****************************************************************************************************************************
 VELOCITY IN Y-DIRECTION
```

```
*************************************************************************************************************************************************/
/*This part calculates the y-velocity inside the actuator disk volume. The y-velocity consists of the y-component of the
tangential velocity and the y-component of the radial velocity.
The tangential velocity is calculated first. Folowed by the computation of the radial velocity*/

DEFINE_PROFILE(y_vel_inboard, t1, indices)
{

 #if !RP_HOST

 /*****************************
 Declaration of local variables
 *****************************/
 /*Mallocing global variables by ensuring that it is only maloced once hence the if statement*/
 /*Variables which are also used globally needs to be malloced first.*/
 if(!VT_z)
 {
  Message0("Allocating VT_z\n");
  VT_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 if(!Vy)
 {
  Message0("Allocating Vy\n");
  Vy = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

  if(!VR_z)
 {
  Message0("Allocating VR_z\n");
  VR_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }
 /*Mallocing global variables by ensuring that it is only malloced once*/

 double VT_y[total_nr_faces_outflowface];
 double VR_y[total_nr_faces_outflowface];
 double r_centroid_flatdisk;
 double ycoord;
 double zcoord;
 double VT;
 double VR;
 double theta;

 double mean_v_Atotal;

 /**Define all array elements related to total_nr_faces_outflowface equal to zero to ensure that computation works in parallel mode**/
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  VT_y[j]=0;
  VR_y[j]=0;
 }
 /**End Define all array elements related to total_nr_faces_outflowface equal to zero to ensure that computation works in
 parallel mode**/

 /*FILE *fileben;*/
 /*********************************
 End Declaration of local variables
 *********************************/
 /**Calculate first the tangential velocity inside the flatdisk**/
 i=0;
 /*fileben = fopen("check_Vtan.dat","w");*/
 begin_c_loop_int(c, t1)
 {
  C_CENTROID(x,c, t1);   /*Calculates the cells' centroid*/
  r_centroid_flatdisk = sqrt(pow((x[1]-yE),2)+pow((x[2]-zE),2));
  ycoord = x[1];
  zcoord = x[2];
  VT = (4*PI*v_A[i]*(VINF+v_A[i]))/(2*PI*r_centroid_flatdisk*OMEGA);
  /**End Calculate first the tangential velocity inside the flatdisk**/

  /**Calculation of radial velocity**/
  mean_v_A =  sum_v_A/total_nr_faces_outflowface;
  VR = (mean_v_A*PI*r_centroid_flatdisk)/(4*RP);
 /* Message("Content of variable mean_v_A is: %d\n", mean_v_A); */
  /**End Calculation of radial velocity**/

   /**Calculation of the angle Theta**/
   if (ycoord!=yE)
   {
    theta = fabs(atan((zcoord-zE)/(ycoord-yE)));
   }
   else
   {
    theta = PI/2;
   }
   /**End Calculation of the angle Theta**/

   /**General Condition for VT_y**/
   if (zcoord<zE)
   {
```

```
    VT_y[i] = pow(-1, sense+1)*VT*sin(theta);
    VR_z[i] = VR*sin(theta);
   }
   if (zcoord>zE)
   {
    VT_y[i] = pow(-1, sense)*VT*sin(theta);
    VR_z[i] = -VR*sin(theta);
   }
   /**End General Condition for VT_y**/

   /**General Condition for VT_z**/
   if (ycoord<yE)
   {
    VT_z[i] = pow(-1, sense)*VT*cos(theta);
    VR_y[i] = VR*cos(theta);
   }

   if (ycoord>yE) /*z<zE & y!=yE*/
   {
    VT_z[i] = pow(-1, sense+1)*VT*cos(theta);
    VR_y[i] = -VR*cos(theta);
   }
   /**End General Condition for VT_z**/
   /**End Calculation of VT_y and VT_z**/

   /**Calculating and imposing the y-velocity**/
   Vy[i] = VT_y[i]+VR_y[i];
   F_PROFILE(c, t1, indices) = Vy[i];
  /*fprintf(fileben,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \n",OMEGA, ycoord, zcoord, theta,
  r_centroid_flatdisk, v_A[i], VT, VT_y[i], VT_z[i], Vy[i]);*/

   i=i+1;
  }
  end_c_loop_int(c, t1)
  /*fclose(fileben);*/
  /**End Calculating and imposing the y-velocity**/

 mean_v_Atotal=PRF_GRSUM1(mean_v_A);
 Message0("Content of mean_v_A is: %f\n", mean_v_Atotal);

  /*Check values*/
  /*Message("Content of counter i is: %d\n", i );

  for (j=0; j<=2*total_nr_faces_outflowface-1; j++)
  {
  Message("Content of VT_y[i] is: %f\n", VT_y[j] );
  /*Message("Content of VT_z[i] is: %f\n", VT_z[j] );*/
  /*Message("Content of V_y[i] is: %f\n", V_y[j] );*/
/*  }*/


   #endif /*!RP_HOST*/

}

/*****************************************************************************************************************************************
End VELOCITY IN Y-DIRECTION
*****************************************************************************************************************************************/


/*****************************************************************************************************************************************
 VELOCITY IN Z-DIRECTION
*****************************************************************************************************************************************/
DEFINE_PROFILE(z_vel_inboard, t1, indices)
{

 #if !RP_HOST

 /*****************************
 Declaration of local variables
 *****************************/
 /*Mallocing global variables by ensuring that it is only malloced once hence the if statement*/
 /*Variables which are also used globally needs to be malloced first.*/
  if(!Vz)
  {
   Message0("Allocating Vz\n");
   Vz = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
  }
 /*End Mallocing global variables by ensuring that it is only malloced once hence the if statement*/

 /**Calculating and imposing the z-velocity**/
 i=0;
 begin_c_loop_int(c, t1)
 {
  Vz[i]=VT_z[i]+VR_z[i];
  F_PROFILE(c, t1, indices) = Vz[i];
  i=i+1;
 }
  end_c_loop_int(c, t1)
  /**End Calculating and imposing the z-velocity**/
```

```
 #endif /*!RP_HOST*/

}
/************************************************************************************************************************************
End VELOCITY IN Z-DIRECTION
************************************************************************************************************************************/
```

B.2 UDF non-uniform actuator disk

```
#include "udf.h"
#include "stdio.h"

/*UDF for M=0.6 & J=3.08 non-uniform Vin (read inflow velocity)*/
/************************************************************************************************************************
Define the constant variables
*****************************************************/
#define VINF 177.91 /* Freestream velocity [m/s].*/
#define omega 92.86 /* Propeller angular velocity (RPS)*/
#define T 197.23 /* proppeller total thrust (N) for one blade was 15.068*/
#define RP 0.311 /* Actuator disk radius [m]*/
#define RLE 0.06 /* Distance from z=0 axis to nacelle contour at x=LE of the blade [m]*/
#define RTE 0.09 /* Distance from z=0 axis to nacelle contour at x=TE of the blade [m]*/
#define x_end 0.215     /*x-coord of the trailing edge of blade [m]*/
#define x_begin 0.13    /*x-coord of the leading edge of blade [m]*/
#define RHO 0.3796 /* Air density at flight Altitude (kg/m^3)*/
#define PI 3.14159265358979323846
#define total_nr_cells_z 42   /* Number of cells in z-direction(=along blade span)*/
#define total_nr_cells_x 13   /* Number of cells in x-direction in blade volume*/
#define x_end_Xtra 0.08   /*x-coord where the extra_volume ends [m] */
#define total_nr_cells_x_Xtra 8 /*Number of cells in x-direction for the extra_volume*/
#define sense 1        /*Sense=1 for clockwise rotating propellers and 0 for counter-clockwise rotating propellers*/
/************************************************************************************************************************/
/*****************************************************/

/************************************************************************************************************************
Declarations of the global variables used in DEFINE_EXECUTE_AFTER_DATA and all the 3 DEFINE_PROFILE progs
************************************************************************************************************************/
/*General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/
Domain *domain;
Thread *t;
face_t f;
cell_t c;
int zone_ID;   /*the term zone_ID is used to move the pointer from one computational zone to another*/
double x[ND_ND];  /*the vector x refers always to the coordinates of the centroid of a face or cell*/
/*End General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/

/*Counters in a loop*/
int i;
int j;
int n;
int teller;
/*End Counters in a loop*/

/*Definition of the disk_inflow and disk_outflow thread pointers*/
Thread *disk_inflow; /*Used to move pointer to the inflow plane of the actuator disk*/
Thread *disk_outflow; /*Used to move pointer to the outflow plane of the actuator disk*/
Thread *extra_volume; /*Used to move pointer to extra_volume*/
Thread *flatdisk; /*Used to move pointer to the flatdisk*/
/*End Definition of the disk_inflow and disk_outflow thread pointers*/

/*Variables needed to define the array sizes*/
int total_nr_faces_inflowface; /*total number of faces of the inflow plane of disk*/
int total_nr_faces_outflowface; /*total number of faces of the outflow plane of disk*/
/*End Variables needed to define the array sizes*/

double *Vin_flatdisk=NULL;
double *U_A_flatdisk=NULL;
double *V_axial_flatdisk=NULL;
double *VT_z=NULL;
double *VR_z=NULL;
double *Vy=NULL;
double *Vz=NULL;

double OMEGA;   /*OMEGA =omega*2PI*/
double Vin[total_nr_cells_z];

double mean_U_A_outflow;
double sum_U_A_outflow;

/************************************************************************************************************************
DEFINE_EXECUTE_AFTER_CASE UDF
************************************************************************************************************************/
/*Use a DEFINE_EXECUTE_AFTER_CASE UDF to calculate the global variables total_nr_faces_inflowface and
total_nr_faces_outflowface in order to prevent mallocing, leading to less memory usage*/
DEFINE_EXECUTE_AFTER_CASE(SR1_flat_disk_final_adj, libudf)
{
 /*Calculation of the inflow velocities of the centroids of the faces of the inflow plane of the blade*/
 domain = Get_Domain(1);
 zone_ID = 30; /*ID_number for the zone of inflow of the blade*/
 disk_inflow = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the inflow domain of the blade*/

 /**Count nr of faces loop**/
 /**This loop runs over all the faces of the inflow plane, to count the total number of faces of the inflow plane of the blade**/
 i=0;
 begin_f_loop(f,disk_inflow)
 {
```

```
 i=i+1;
}
end_f_loop(f,disk_inflow)

total_nr_faces_inflowface=i;

Message("Content of variable total_nr_faces_inflowface is: %d\n", total_nr_faces_inflowface);
/**End Count nr of faces loop**/
/**End Calculation of the number of faces of the inflow plane of the blade**/


/**Calculation of the number of faces of the outflow plane of the actuator disk**/
zone_ID = 28; /*ID_number for the zone of outflow of the actuator disk*/
disk_outflow = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the outflow domain of the actuator disk*/

/**Count nr of faces loop**/
/**This loop runs over all the faces of the outflow plane, to count the total number of faces of the outflow plane of the disk**/
i=0;
begin_f_loop(f,disk_outflow)
{
 i=i+1;
}
end_f_loop(f,disk_outflow)

total_nr_faces_outflowface=i;

Message("Content of variable total_nr_faces_outflowface is: %d\n", total_nr_faces_outflowface);
/**End Count nr of faces loop**/
/**End Calculation of the number of faces of the outflow plane of the actuator disk**/

/**Specify thread of the extra_volume to read the inflow velocities**/
zone_ID = 13; /*ID_number for the zone of the extra_volume*/
extra_volume = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the extra_volume*/
/**End Specify thread of the extra_volume to read the inflow velocities**/

}
/************************************************************************************************************************
End DEFINE_EXECUTE_AFTER_CASE UDF
************************************************************************************************************************/


/************************************************************************************************************************
 VELOCITY IN X-DIRECTION
************************************************************************************************************************/
DEFINE_PROFILE(axial_velocity, t1, indices)
{
 /*****************************
 Declaration of local variables
 *****************************/
 if(!V_axial_flatdisk)
 {
  Message("Allocating V_axial_flatdisk\n");
  V_axial_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

  if(!U_A_flatdisk)
 {
  Message("Allocating U_A_flatdisk\n");
  U_A_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 if(!Vin_flatdisk)
 {
  Message("Allocating Vin_flatdisk\n");
  Vin_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 /**Define the import variables from matlab files**/
 /*Polynomial coefficients C1 & C2 describing the local pathlines C1*x+C2*/
 float C_1[total_nr_cells_z];
 float C_2[total_nr_cells_z];
 FILE *inputFileC2;
 FILE *inputFileC1;
 /**End Define the import variables from matlab files**/

 /**Define variables belonging to the inflow_face of the blade**/
 double criterium;
 double r_xtra;
 double x_xtra;
 double u_inflow;
 double v_inflow;
 double w_inflow;

 double r_inflow_centroid[total_nr_faces_inflowface]; /*Radius of the centroid of the faces of inflow plane of actuator*/
 double V_axial_centroid_in[total_nr_faces_inflowface]; /*Velocity of the centroid of the faces of inflow plane of actuator*/
 int rnew_in[total_nr_faces_inflowface];
 double checkpoint_in;
 int newcheckpoint_in;
 int positie_in;
 int teller_in;
```

```
double inflow_centroid_r[total_nr_faces_inflowface];
double inflow_centroid_V[total_nr_faces_inflowface];
double old_inflow_centroid_r[total_nr_faces_inflowface];
double old_inflow_centroid_V[total_nr_faces_inflowface];
int size_inflow_array;
double max_inflow_centroid_r;
double min_inflow_centroid_r;
double arranged_inflow_centroid_r[total_nr_faces_inflowface];
double arranged_inflow_centroid_V[total_nr_faces_inflowface];
int position_min;
double old_positionvector[total_nr_faces_inflowface];
int maximum_position;
int maximum_pos[total_nr_faces_inflowface];
int max_pos;
/**End Define variables belonging to the inflow_face of the blade**/

/**Define variables belonging to the flatdisk**/
double r_outflow_centroid[total_nr_faces_outflowface];
double saved_outflow_centroid_r[total_nr_faces_outflowface];
double xcoord[total_nr_faces_outflowface];
/*double ycoord[total_nr_faces_outflowface];
double zcoord[total_nr_faces_outflowface];*/

double dr_end;

int r_outnew[total_nr_faces_outflowface]; /*Integer to ensure the needed significant digits*/
double checkpoint_out;
int newcheckpoint_out;
int positie_out;
int teller_out;
double outflow_centroid_r[total_nr_faces_outflowface];
double old_outflow_centroid_r[total_nr_faces_outflowface];
int size_outflow_array;
double max_outflow_centroid_r;
double min_outflow_centroid_r;
double arranged_outflow_centroid_r[total_nr_faces_outflowface];
int position_min_out;
double old_positionvector_out[total_nr_faces_outflowface];
int maximum_position_out;
int maximum_pos_out[total_nr_faces_outflowface];

double r_polynomial[total_nr_cells_z];
int index;
double lower_r_polynomial;
double upper_r_polynomial;
double differ_polynomial;
double dif_poly_centroid;
double fraction_poly;
double r_begin[total_nr_cells_z];

double r_lower ;
double V_lower;
double r_upper;
double V_upper;
double dif_r;
double ref_r;
double perc_r;

double V_i;
double Vxs;
double omega_s;
double drs;
double rs[total_nr_cells_z];
double Rs;
double difference_p[total_nr_cells_z];
double dT_centroid[total_nr_cells_z];
double check_thrust1;

double VT_inf[total_nr_cells_z];
double U_A_outflow[total_nr_cells_z];
double V_axial[total_nr_cells_z];

int r_new[total_nr_cells_z]; /*Integer to ensure the needed significant digits*/
int newcheckpoint[total_nr_faces_outflowface];
int positie;

/*FILE *file1;
FILE *file2;
FILE *file3;
FILE *file4;
FILE *file5;
FILE *file6;
FILE *file7;
FILE *file8;
FILE *filedT;
FILE *file9;
FILE *file10;
FILE *file11;
FILE *file12;*/
```

```
/**End Define variables belonging to the flatdisk**/
/*********************************
End Declaration of local variables
*********************************/

/**Import variables from matlab files**/
inputFileC1 = fopen("pathline_C1.txt", "rb");
i=0;
while(!feof(inputFileC1))
{
 fscanf(inputFileC1, "%f", &C_1[i]);
 i++;
}
fclose(inputFileC1);


inputFileC2 = fopen("pathline_C2.txt", "rb");
i=0;
while(!feof(inputFileC2))
{
 fscanf(inputFileC2, "%f", &C_2[i]);
 i++;
}
fclose(inputFileC2);
/**End Import variables from matlab files**/

/**Calculate r_inflow_centroid and the inflow velocities of the centroids, V_axial_centroid_in.**/
/*file1 = fopen("check1.dat","w");*/
criterium = x_begin-((x_begin-x_end_Xtra)/total_nr_cells_x_Xtra);
j=0;
n=0;
begin_c_loop(c, extra_volume)
{
 C_CENTROID(x,c,extra_volume);   /*Calculates the cells' centroid*/
 r_xtra =sqrt((x[1])*(x[1])+(x[2])*(x[2]));  /*Radius computation of centroid of the cells of iextra_volume*/
 x_xtra = x[0];
 if (x_xtra > criterium) /*Find the last column of extra_volume*/
 {
  r_inflow_centroid[n]=sqrt((x[1])*(x[1])+(x[2])*(x[2]));  /*Radius computation of centroid at cells of the extra_volume*/
  u_inflow=C_U(c,extra_volume);  /*Read the x, y and z components of inflow velocity using C_U, C_V and C_W.*/
  v_inflow=C_V(c,extra_volume);
  w_inflow=C_W(c,extra_volume);
  V_axial_centroid_in[n]= sqrt(pow(u_inflow,2)+pow(v_inflow,2)+pow(w_inflow,2));  /*V=sqrt(u^2+v^2+w^2)*/
  /*fprintf(file1,"%.6g \t %.6g \n",r_inflow_centroid[n], V_axial_centroid_in[n]);*/
  n=n+1;
 }
 j=j+1;
}
end_c_loop(c, extra_volume)
Message("Content of variable criterium is: %f\n", criterium);
Message("Content of variable n should be equal to total_nr_faces_inflowface and is: %d\n", n);
/**n should be equal to total_nr_faces_inflowface**/
/*fclose(file1);*/
/**End Calculate r_inflow_centroid and the inflow velocities of the centroids, V_axial_centroid_in.**/

/**It is sufficient to define the r_inflow_centroid until 4 digits after the comma. In order to isolate the same
r_inflow_centroid from the total array, define an int which is (1000* r_inflow_centroid). Do the same thing for
the reference values checkpoint and call the corresponding
integers newcheckpoint*/
/**Define new integer rnew to ensure the needed significant digits**/
for (j=0; j<=total_nr_faces_inflowface-1; j++)
{
 rnew_in[j] =pow(10,3)*r_inflow_centroid[j];
}
/**Define new integer rnew to ensure the needed significant digits**/
/**Loop to extract the same r_inflow_centroid from the array and declaration of the corresponding inflow velocities**/
for (j=0; j<=total_nr_faces_inflowface-1; j++)
{
 for (n=j+1; n<=total_nr_faces_inflowface-1; n++)
 {
  checkpoint_in = r_inflow_centroid[j];
  newcheckpoint_in=pow(10,3)*checkpoint_in;
  if (newcheckpoint_in == rnew_in[n])
  {
   if(newcheckpoint_in!=0)
   {
    positie_in = n;
    r_inflow_centroid[positie_in]= 0;
    V_axial_centroid_in[positie_in] = 0;
   }
  }
 }
}
/**Loop to extract the same r_inflow_centroid from the array and declaration of the corresponding inflow velocities**/

/**Count how many r_inflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/
teller_in=0;
for (j=0; j<=total_nr_faces_inflowface-1; j++)
{
 if (r_inflow_centroid[j]!=0)
```

```
 {
  teller_in=teller_in+1;
 }
}

Message("teller_face is: %d\n", teller_in);
/*for (j=0; j<total_nr_faces_inflowface-1; j++)
{
Message("Content of variable  r_inflow_centroid is: %f\n",  r_inflow_centroid[j]);
}
/**End Count how many r_inflow_centroid are non-zero.**/


/**Gather all non-zero elements of r_inflow_centroid in one single array called new_r_inflow_centroid**/
/**Gather all non-zero elements of V_axial_centroid_in in one single array called V_in_centroid**/
 /*file2 = fopen("check2.dat","w");*/
 i=0;
 for (j=0; j<=total_nr_faces_inflowface-1; j++)
 {
  if (r_inflow_centroid[j]!=0)
  {
   inflow_centroid_r[i] = r_inflow_centroid[j];
   old_inflow_centroid_r[i] = inflow_centroid_r[i];
   inflow_centroid_V[i] = V_axial_centroid_in[j];
   old_inflow_centroid_V[i] = inflow_centroid_V[i];
   /*fprintf(file2,"%.6g \t %.6g \t %.6g \t %.6g \n",old_inflow_centroid_r[i], inflow_centroid_r[i],
   inflow_centroid_V[i], old_inflow_centroid_V[i]);*/
   i=i+1;
  }
 }
 size_inflow_array = i;
 /*fclose(file2);*/

 Message("size_inflow_array is: %d\n", size_inflow_array);
 /*for (j=0; j<=teller-1; j++)
 {
  Message("Content of inflow_centroid_r is: %f\n",  inflow_centroid_r[j]);
 }*/
/**End Gather all non-zero elements of r_inflow_centroid in one single array called new_r_inflow_centroid**/
/**End Gather all non-zero elements of V_axial_centroid_in in one single array called V_in_centroid**/
/**Arrange the values of r_inflow_centroid from small to large and find the corresponding Velocity**/
/*file3 = fopen("check3.dat","w");*/
for (j= 0; j<=size_inflow_array-1; j++)
{
 i=1;
 /**Find maximum value of array inflow_centroid_r**/
 max_inflow_centroid_r = 0;
 for (n=0; n<=size_inflow_array-1; n++)
 {
  if (inflow_centroid_r[n]> max_inflow_centroid_r)
  {
   max_inflow_centroid_r = inflow_centroid_r[n];
   maximum_position = n;
  }
 }
 maximum_pos[j] = maximum_position;
 /**End Find maximum value of array inflow_centroid_r**/
 /**Find minimum value of array r_random**/
 min_inflow_centroid_r = max_inflow_centroid_r;
 for (n=0; n<=size_inflow_array-1; n++)
 {
  if (inflow_centroid_r[n]< min_inflow_centroid_r)
  {
   min_inflow_centroid_r = inflow_centroid_r[n];
   position_min = n;
  }
 }
 /**End Find minimum value of array r_random**/
  old_positionvector[j]= position_min;
  arranged_inflow_centroid_V[j] = old_inflow_centroid_V[position_min];

 /**Arrange inflow_centroid_r from small to large**/
 arranged_inflow_centroid_r[j] = min_inflow_centroid_r;
 /**Replace inflow_centroid_r[position_min] with max_inflow_centroid_r so that we can find the next minimum**/
 inflow_centroid_r[position_min]= max_inflow_centroid_r;
 /*fprintf(file3,"%.6g \t %.6g \t %.6g \t %.6g \n",old_inflow_centroid_r[j], arranged_inflow_centroid_r[j],
 old_inflow_centroid_V[j], arranged_inflow_centroid_V[j]);*/
}
/*fclose(file3);*/
 /**The last entry of arranged_inflow_centroid_r will be the max of old_inflow_centroid_r. So the last entry
 of arranged_inflow_centroid_V must be the velocity that corresponds to the max of old_inflow_centroid_r, which
 position is given by the first entry of maximum_pos, which stores the position values of the maxima**/
 max_pos = maximum_pos[0];
 arranged_inflow_centroid_V[total_nr_cells_z-1] = old_inflow_centroid_V[max_pos];
 Message("Content of last arranged_inflow_centroid_V is: %f\n",  arranged_inflow_centroid_V[total_nr_cells_z-1]);
 /*Check of the answers*/
 /*for (j=0; j<=size_inflow_array-1; j++)
 {
  Message("Content of  arranged_inflow_centroid_r is: %f\n",  arranged_inflow_centroid_r[j]);
  Message("Content of  arranged_inflow_centroid_V is: %f\n",  arranged_inflow_centroid_V[j]);
```

```
 }
 for (j=0; j<=size_inflow_array-1; j++)
 {
  Message("Content of old_inflow_centroid_r is: %f\n",  old_inflow_centroid_r[j]);
  Message("Content of old_inflow_centroid_V is: %f\n",  old_inflow_centroid_V[j]);
 }
 /*End Check of the answers*/
/**End Arrange the values of r_inflow_centroid from small to large and find the corresponding Velocity**/

 /**Calculate the radii of the centroids of flatdisk**/
 zone_ID = 14; /*ID_number for the zone of the flatdisk*/
 flatdisk = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the flatdisk domain*/
 dr_end =(RP-RTE)/total_nr_cells_z;  /*Calculation of the elemental width dr_end*/
 OMEGA = 2*PI*omega;
 /*file4 = fopen("check4.dat","w");*/
 i=0;
 begin_c_loop(c, flatdisk)
 {
  C_CENTROID(x,c,flatdisk);   /*Calculates the cell's centroid*/
  r_outflow_centroid[i]=sqrt((x[1])*(x[1])+(x[2])*(x[2]));  /*Radius computation of centroid of cells of the flatdisk*/
  saved_outflow_centroid_r[i] = r_outflow_centroid[i]; /*We need this to now the V_axial_flatdisk for each cell
  within flatdisk*/
  xcoord[i] = x[0];
  /*ycoord[i] = x[1];
  zcoord[i] = x[2];*/
 /* fprintf(file4," %.6g \n",r_outflow_centroid[i]);*/
  i=i+1;
 }
 end_c_loop(c, flatdisk)
/* fclose(file4);*/
 /**End Calculate the radii of the centroids of flatdisk**/

 /**It is sufficient to define the r_outflow_centroid until 3 digits after the comma. In order to isolate the same
 r_outflow_centroid from the total array, define an int which is (1000* r_outflow_centroid). Do the same thing
 for the reference values checkpoint and call the corresponding integers newcheckpoint*/
 /**Define new integer rnew to ensure the needed significant digits**/
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  r_outnew[j] =pow(10,3)*r_outflow_centroid[j];
 }
 /**End Define new integer r_outnew to ensure the needed significant digits**/
 /**Loop to extract the same r_outflow_centroid from the array**/
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  for (n=j+1; n<=total_nr_faces_outflowface-1; n++)
  {
   checkpoint_out = r_outflow_centroid[j];
   newcheckpoint_out=pow(10,3)*checkpoint_out;
   if (newcheckpoint_out == r_outnew[n])
   {
    if(newcheckpoint_out!=0)
    {
     positie_out = n;
     r_outflow_centroid[positie_out]= 0;
    }
   }
  }
 }
 /**Loop to extract the same r_outflow_centroid from the array and declaration of the corresponding outflow velocities**/

 /**Count how many r_outflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/
 teller_out=0;
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  if (r_outflow_centroid[j]!=0)
  {
   teller_out=teller_out+1;
  }
 }
 Message("teller_out is: %d\n", teller_out);
 /**End Count how many r_outflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/

 /**Gather all non-zero elements of r_outflow_centroid in one single array called new_r_outflow_centroid**/
 /*file5= fopen("check5.dat","w");*/
 i=0;
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  if (r_outflow_centroid[j]!=0)
  {
   outflow_centroid_r[i] = r_outflow_centroid[j];
   old_outflow_centroid_r[i] = outflow_centroid_r[i];
   /*fprintf(file5,"%.6g \t %.6g \n",outflow_centroid_r[i], old_outflow_centroid_r[i]);*/
   i=i+1;
  }
 }
 size_outflow_array = i;
/* fclose(file5);*/

 Message("size_outflow_array is: %d\n", size_outflow_array);
 /*for (j=0; j<=teller_out-1; j++)
```

```
 {
  Message("Content of outflow_centroid_r is: %f\n",  outflow_centroid_r[j]);
 }*/
 /**End Gather all non-zero elements of r_outflow_centroid in one single array called new_r_outflow_centroid**/

 /**Arrange the values of r_outflow_centroid from small to large**/
 /*file6= fopen("check6.dat","w");*/
 for (j= 0; j<=size_outflow_array-1; j++)
 {
  i=1;
  /**Find maximum value of array inflow_centroid_r**/
  max_outflow_centroid_r = 0;
  for (n=0; n<=size_outflow_array-1; n++)
  {
   if (outflow_centroid_r[n]> max_outflow_centroid_r)
   {
    max_outflow_centroid_r = outflow_centroid_r[n];
    maximum_position_out = n;
   }
  }
  maximum_pos_out[j] = maximum_position_out;
  /**End Find maximum value of array outflow_centroid_r**/
  /**Find minimum value of array r_random**/
  min_outflow_centroid_r = max_outflow_centroid_r;
  for (n=0; n<=size_outflow_array-1; n++)
  {
   if (outflow_centroid_r[n]< min_outflow_centroid_r)
   {
    min_outflow_centroid_r = outflow_centroid_r[n];
    position_min_out = n;
   }
  }
  /**End Find minimum value of array r_random**/
  old_positionvector_out[j]= position_min_out;

  /**Arrange outflow_centroid_r from small to large**/
  arranged_outflow_centroid_r[j] = min_outflow_centroid_r;
  /**Replace outflow_centroid_r[position_min] with max_outflow_centroid_r so that we can find the next minimum**/
  outflow_centroid_r[position_min_out]= max_outflow_centroid_r;
  /*fprintf(file6,"%.6g \t %.6g \n", old_outflow_centroid_r[j], arranged_outflow_centroid_r[j]);*/
 }
/* fclose(file6);*/
 /*Check of the answers*/
 /*for (j=0; j<=size_outflow_array-1; j++)
 {
  Message("Content of  arranged_outflow_centroid_r is: %f\n",  arranged_outflow_centroid_r[j]);
 }
 for (j=0; j<=size_outflow_array-1; j++)
 {
  Message("Content of old_outflow_centroid_r is: %f\n",  old_outflow_centroid_r[j]);
 }*/
 /*End Check of the answers*/
 /**End Arrange the values of r_outflow_centroid from small to large and find the corresponding Velocity**/

 /**Calculate the r_begin corresponding to arranged_outflow_centroid_r**/
 /*file7= fopen("check7.dat","w");*/
for (i=0; i<=total_nr_faces_outflowface-1; i++)
{
 for (j=0; j<=total_nr_cells_z-1; j++)
 {
  r_polynomial[j] = C_1[j]*xcoord[i]+C_2[j];
 }
}
for(i=0; i<=total_nr_cells_z-1;i++)
{
 for(j=0; j<=total_nr_cells_z-2;j++)
 {
  if(arranged_outflow_centroid_r[i]>=r_polynomial[j] && arranged_outflow_centroid_r[i]<=r_polynomial[j+1])
  {
   index=j;
  }
  else
  {
   index=-1;
  }
  if (index!=-1)
  {
   lower_r_polynomial = r_polynomial[index];
   upper_r_polynomial = r_polynomial[index+1];
   differ_polynomial=upper_r_polynomial-lower_r_polynomial;
   dif_poly_centroid = upper_r_polynomial-arranged_outflow_centroid_r[i];
   fraction_poly = dif_poly_centroid/differ_polynomial;
   r_begin[i]=(C_1[index+1]*(1-fraction_poly)+C_1[index]*fraction_poly)*x_begin+(C_2[index+1]*(1-fraction_poly)+
   C_2[index]*fraction_poly);
  }
  else
  {
   if(arranged_outflow_centroid_r[i]<=r_polynomial[0])
   {
    r_begin[i]=C_1[0]*x_begin + C_2[0];
```

```
    }
    if(arranged_outflow_centroid_r[i]>=r_polynomial[total_nr_cells_z-1])
    {
     r_begin[i]=C_1[total_nr_cells_z-1]*x_begin + C_2[total_nr_cells_z-1];
    }
   }
  }
  /*fprintf(file7,"%.6g \t %.6g \t %.6g \t %.6g \n", lower_r_polynomial, arranged_outflow_centroid_r[i],
  upper_r_polynomial, r_begin[i]);*/
 }
  /*fclose(file7);*/
 /**End Calculate the r_begin corresponding to arranged_outflow_centroid_r**/

  /**Find Vin corresponding to r_begin[i]**/
  /*file8= fopen("check8.dat","w");*/
  for (j=0; j<=total_nr_cells_z-1; j++)
   {
    if(r_begin[j]<= arranged_inflow_centroid_r[0])
    {
     Vin[j]=arranged_inflow_centroid_V[0];
    }

    if(r_begin[j]>= arranged_inflow_centroid_r[total_nr_cells_z-1])
    {
     Vin[j]=arranged_inflow_centroid_V[total_nr_cells_z-1];
    }
    for (n=0; n<=total_nr_cells_z-2; n++)
    {
     if(r_begin[j]>= arranged_inflow_centroid_r[n] && r_begin[j]<= arranged_inflow_centroid_r[n+1])
     {
      r_lower = arranged_inflow_centroid_r[n];
      V_lower=arranged_inflow_centroid_V[n];
      r_upper = arranged_inflow_centroid_r[n+1];
      V_upper=arranged_inflow_centroid_V[n+1];
      dif_r= r_upper-r_lower;
      ref_r = r_upper-r_begin[j];
      perc_r=ref_r/dif_r;
      Vin[j] = V_upper*(1-perc_r)+V_lower*perc_r;
      /*fprintf(file8,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \t  %.6g \n",r_lower, r_begin[j], r_upper, V_lower,
      Vin[j], V_upper);*/
      }
     }
    }
   /*fclose(file8);*/
   /**End Find Vin corresponding to r_begin[i]**/


   /**Now that we have total_nr_cells_z non-zero arranged_outflow_centroid_r arranged from small to large and
   corresponding Vin, one can determine the elemental thrust for these elements.*/
   /*filedT = fopen("dT.dat","w");*/
   for (i=0; i<=total_nr_cells_z-1; i++)
   {
    V_i=sqrt((pow(VINF,2)/4)+((pow(OMEGA,2)*pow(RP,2))/4)*
    (1-sqrt(1-((4*T)/(PI*(pow(RP,2)-pow(RTE,2))*RHO*pow(OMEGA,2)*pow(RP,2))))))-(VINF/2);
    Vxs = VINF+2*V_i;
    omega_s = (4*(VINF+2*V_i)*V_i)/(OMEGA*pow(RP,2));
    drs =sqrt((VINF+V_i)/Vxs)*dr_end;  /*Calculation of the elemental width drs*/
    rs[i] =sqrt(((VINF+V_i)/Vxs)*(pow(arranged_outflow_centroid_r[i],2)));
    Rs =sqrt((VINF+V_i)/Vxs)*RP;
    difference_p[i] = -(RHO*pow(omega_s,2)/2)*(pow(Rs,2)-pow(rs[i],2));
    dT_centroid[i] =(Vxs-VINF)*RHO*Vxs*((2*PI)*rs[i]*drs)+difference_p[i]*((2*PI)*rs[i]*drs); /*Elemental thrust of annular element*/
    /*Check values. Make sure that the root is taken of a positive number*/
    /*sqrt_Vi= 1-((4*T)/(PI*(pow(RP,2)-pow(RTE,2))*RHO*pow(OMEGA,2)*pow(RP,2)));*/
   /* fprintf(filedT,"%.6g \t %.6g \n", arranged_outflow_centroid_r[i], dT_centroid[i]);*/
   }
   /*fclose(filedT);*/
   /*End Loop to calculate the elemental thrust of the centroids of the outflow plane (dT_centroid)*/

   /*Check whether the thrust calculation is correct. The total thrust is divided over all the individual faces of
   the outflow plane.Since we have total_nr_cells_z different outflow_centroid_r, we will have total_nr_cells_z elemental
   thrusts. The total thrust is equal to the sum of all the elemental thrusts of each face of the outflow plane.
   So the sum of all dT_centroid*(total_nr_faces_outflowface/total_nr_cells_z) must equal the total Thrust T*/
   check_thrust1 = dT_centroid[0];

   for (i=0; i<=total_nr_cells_z-2; i++)
   {
   check_thrust1 = check_thrust1 + dT_centroid[i+1];
   check_thrust1 = check_thrust1;
   }

   /*Check whether the thrust calculation is correct. All dT_centroids must be positive*/
   j=0;
   for (i=0; i<=total_nr_cells_z-1; i++)
   {
    if (dT_centroid[i]>=0)
{
 j=j+1;
}
   }
   Message("Content of check_thrust1 is: %f\n", check_thrust1);
```

```
  Message("Content of j is: %d\n", j );
  /*End Check whether the thrust calculation is correct. The sum of all dT_centroid must equal T*/

  /**Calculation of the axial velocity at flatdisk**/
  /* file9= fopen("check9.dat","w");*/
   for(i=0; i<=total_nr_cells_z-1; i++)
  {
   VT_inf[i] = omega_s*rs[i];
   U_A_outflow[i] = 0.5*(-Vin[i] + sqrt(pow(Vin[i],2) + pow(VT_inf[i],2) +
   dT_centroid[i]/(RHO*PI*arranged_outflow_centroid_r[i]*dr_end)));
   V_axial[i]= Vin[i] +U_A_outflow[i];

   /*fprintf(file9,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \n", arranged_outflow_centroid_r[i], dT_centroid[i], VT_inf[i],
    U_A_outflow[i],V_axial[i]);*/
  }
   sum_U_A_outflow = U_A_outflow[0];

   for (i=0; i<=total_nr_cells_z-2; i++)
   {
    sum_U_A_outflow = sum_U_A_outflow + U_A_outflow[i+1];
    sum_U_A_outflow = sum_U_A_outflow;
   }

  /*fclose(file9);/*
  /**End Calculation of the axial velocity at flatdisk**/

  /**Find V_axial_flatdisk for total_nr_faces_outflowface number of cells**/
  /**The axial velocity is now known for total_nr_cells_z, which is axisymmetric so now we have to impose the right V_axial to
  the right saved_outflow_centroid_r. the array saved_outflow_centroid_r must be used since the original r_outflow_centroid is
  changed because of the sorting prog**/
  /**It is sufficient to define the arranged_r_outflow_centroid until 3 digits after the comma. In order to isolate the same
  saved_outflow_centroid_r from the total array, define an int which is (1000* arranged_r_outflow_centroid). Do the same thing
  for the reference values checkpoint and call the corresponding integers newcheckpoint*/
  /**Define new integer rnew to ensure the needed significant digits**/
 /* file10= fopen("check10.dat","w");*/
  for (j=0; j<=total_nr_cells_z-1; j++)
  {
   r_new[j] =pow(10,3)*arranged_outflow_centroid_r[j];
   /*fprintf(file10," %.6g \n",arranged_outflow_centroid_r[j]);*/
  }
  /*fclose(file10);*/

   /*file11= fopen("check11.dat","w");*/
   for (n=0; n<=total_nr_faces_outflowface-1; n++)
  {
   newcheckpoint[n] =pow(10,3)*saved_outflow_centroid_r[n];
   /*fprintf(file11," %.6g \n", saved_outflow_centroid_r[n]); */
  }
  /*fclose(file11);*/
  /**End Define new integer r_outnew to ensure the needed significant digits**/
  /**Loop to extract the same r_outflow_centroid from the array**/

  for (j=0; j<=total_nr_cells_z-1; j++)
  {
   for (n=0; n<=total_nr_faces_outflowface-1; n++)
   {
    if (r_new[j] == newcheckpoint[n])
    {
     if(newcheckpoint[n]!=0)
     {
      positie = n;
      V_axial_flatdisk[positie]= V_axial[j];
      Vin_flatdisk[positie]= Vin[j];
      U_A_flatdisk[positie] = U_A_outflow[j];
     }
    }
   }
  }

/*  file12= fopen("check12.dat","w");
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
  {
  fprintf(file12,"%.6g \t %.6g \n", V_axial_flatdisk[j], Vin_flatdisk[j]);
  }
  fclose(file12);*/

  /**End Find V_axial_flatdisk for total_nr_faces_outflowface number of cells**/
  /**Impose the axial velocity using F_PROFILE**/
  i=0;
  begin_c_loop(c, t1)
  {
   F_PROFILE(c, t1, indices) = V_axial_flatdisk[i];
   i=i+1;
  }
  end_c_loop(c, t1)
  /**End Impose the axial velocity using F_PROFILE**/

}
/*****************************************************************************************************************
End VELOCITY IN X-DIRECTION
```

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


/*****************************************************************************************************************************
 VELOCITY IN Y-DIRECTION
*****************************************************************************************************************************/
/*This part calculates the y-velocity inside the actuator disk volume. The y-velocity consists of the y-component of
the tangential velocity and the y-component of the radial velocity.
The tangential velocity is calculated first. Folowed by the computation of the radial velocity*/

DEFINE_PROFILE(y_velocity, t1, indices)
{
 if(!VT_z)
 {
  Message("Allocating VT_z\n");
  VT_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 if(!Vy)
 {
  Message("Allocating Vy\n");
  Vy = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

  if(!VR_z)
 {
  Message("Allocating VR_z\n");
  VR_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 double VT_y[total_nr_faces_outflowface];
 double VR_y[total_nr_faces_outflowface];

 /*******************************
 Declaration of local variables
 *******************************/
 double r_centroid_flatdisk;
 double ycoord;
 double zcoord;
 double VT;
 double VR;
 double theta;

 /*FILE *fileben;*/
 /***********************************
 End Declaration of local variables
 ***********************************/
 /**Calculate first the tangential velocity inside the flatdisk**/
 i=0;
 /*fileben = fopen("check_Vtan.dat","w");*/
 begin_c_loop(c, t1)
 {
  C_CENTROID(x,c, t1);   /*Calculates the cells' centroid*/
  r_centroid_flatdisk = sqrt((x[1])*(x[1])+(x[2])*(x[2]));
  ycoord = x[1];
  zcoord = x[2];
  VT = (4*PI*U_A_flatdisk[i]*(Vin_flatdisk[i]+U_A_flatdisk[i]))/(2*PI*r_centroid_flatdisk*OMEGA);
  /**End Calculate first the tangential velocity inside the flatdisk**/

  /**Calculation of radial velocity**/
  mean_U_A_outflow =  sum_U_A_outflow/total_nr_cells_z;
  VR =  (mean_U_A_outflow*PI*r_centroid_flatdisk)/(4*RP);
  /**End Calculation of radial velocity**/

   /**Calculation of the angle Theta**/
   if (ycoord!=0)
   {
    theta = fabs(atan(zcoord/ycoord));
   }
   else
   {
    theta = PI/2;
   }
   /**End Calculation of the angle Theta**/

   /**General Condition for VT_y**/
   if (zcoord<0)
   {
    VT_y[i] = pow(-1, sense+1)*VT*sin(theta);
    VR_z[i] = VR*sin(theta);
   }
   if (zcoord>0)
   {
    VT_y[i] = pow(-1, sense)*VT*sin(theta);
    VR_z[i] = -VR*sin(theta);
   }
   /**End General Condition for VT_y**/

   /**General Condition for VT_z**/
   if (ycoord<0)
```

```
   {
    VT_z[i] = pow(-1, sense)*VT*cos(theta);
    VR_y[i] = VR*cos(theta);
   }

   if (ycoord>0) /*z<0 & y!=0*/
   {
    VT_z[i] = pow(-1, sense+1)*VT*cos(theta);
    VR_y[i] = -VR*cos(theta);
   }
   /**End General Condition for VT_z**/
   /**End Calculation of VT_y and VT_z**/

   Vy[i] = VT_y[i]+VR_y[i];
   F_PROFILE(c, t1, indices) = Vy[i];
  /*fprintf(fileben,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \n",OMEGA, ycoord,
   zcoord, theta, r_centroid_flatdisk, Vin_flatdisk[i], V_axial_flatdisk[i], VT, VT_y[i], VT_z[i]);*/

   i=i+1;
  }
  end_c_loop(c, t1)
  /*fclose(fileben);*/

  /**End Calculation of Vr_y and Vr_z**/

/*   Message("Content of counter i is: %d\n", i );
   Message("Content of counter teller is: %d\n", teller );
   Message("Content of counter teller2 is: %d\n", teller2 );



  for (j=0; j<=2*total_nr_cells_z-1; j++)
  {
  Message("Content of VT_y[i] is: %f\n", VT_y[j] );
  /*Message("Content of VT_z[i] is: %f\n", VT_z[j] );*/
  /*Message("Content of V_y[i] is: %f\n", V_y[j] );*/
/*  }*/

}

/*****************************************************************************************************************
End VELOCITY IN Y-DIRECTION
*****************************************************************************************************************/


/*****************************************************************************************************************
 VELOCITY IN Z-DIRECTION
*****************************************************************************************************************/
DEFINE_PROFILE(z_velocity, t1, indices)
{
  if(!Vz)
  {
   Message("Allocating Vz\n");
   Vz = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
  }


 i=0;
 begin_c_loop(c, t1)
 {
  Vz[i]=VT_z[i]+VR_z[i];
  F_PROFILE(c, t1, indices) = Vz[i];
  i=i+1;
 }
  end_c_loop(c, t1)
}
/*****************************************************************************************************************
End VELOCITY IN Z-DIRECTION
*****************************************************************************************************************/
```

B.3 UDF Phillips' equations

```
#include "udf.h"
#include "stdio.h"

/*UDF for M=0.6 & J=3.08 uniform Vin equal to freestream using Phillips' equations*/
/*****************************************************************************************************************
Define the constant variables
*****************************************************/
#define VINF 177.91 /* Freestream velocity [m/s].*/
#define omega 92.86 /* Propeller angular velocity (RPS)*/
#define T 197.23 /* proppeller total thrust (N) for one blade was 24.654*/
#define RP 0.311 /* Actuator disk radius [m]*/
#define RLE 0.06 /* Distance from z=0 axis to nacelle contour at x=LE of the blade [m]*/
#define RTE 0.09 /* Distance from z=0 axis to nacelle contour at x=TE of the blade [m]*/
```

```
#define x_end 0.215     /*x-coord of the trailing edge of blade [m]*/
#define x_begin 0.13    /*x-coord of the leading edge of blade [m]*/
#define RHO 0.3796 /* Air density at flight Altitude (kg/m^3)*/
#define PI 3.14159265358979323846
#define total_nr_cells_z 42   /* Number of cells in z-direction(=along blade span)*/
#define total_nr_cells_x 14   /* Number of cells in x-direction*/
#define arc_cells 40   /* Number of cells in along the actuator's arc. It should normally be equal to the total_nr_cells_z */
#define sense 1        /*Sense=1 for clockwise rotating propellers and 0 for counter-clockwise rotating propellers*/
/*****************************************************************************************************************************/
/*****************************************************/

/*****************************************************************************************************************************
Declarations of the global variables used in DEFINE_EXECUTE_AFTER_DATA and all the 3 DEFINE_PROFILE progs
*****************************************************************************************************************************/
/*General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/
Domain *domain;
Thread *t;
face_t f;
cell_t c;
int zone_ID;   /*the term zone_ID is used to move the pointer from one computational zone to another*/
double x[ND_ND];  /*the vector x refers always to the coordinates of the centroid of a face or cell*/
/*End General domain variables to define the domain, threat, face, cell, zone_ID and centroids*/

/*Counters in a loop*/
int i;
int j;
int n;
int teller;
/*End Counters in a loop*/

/*Definition of the disk_inflow and disk_outflow thread pointers*/
Thread *disk_inflow; /*Used to move pointer to the inflow plane of the actuator disk*/
Thread *disk_outflow; /*Used to move pointer to the outflow plane of the actuator disk*/
Thread *flatdisk; /*Used to move pointer to the flatdisk*/
/*End Definition of the disk_inflow and disk_outflow thread pointers*/

/*Variables needed to define the array sizes*/
int total_nr_faces_inflowface; /*total number of faces of the inflow plane of disk*/
int total_nr_faces_outflowface; /*total number of faces of the outflow plane of disk*/
/*End Variables needed to define the array sizes*/

double V_i;
double Vxs;
double omega_s;
double *rs=NULL;
double *Vin_flatdisk=NULL;
double *U_A_flatdisk=NULL;
double *V_axial_flatdisk=NULL;
double *VT_z=NULL;
/*double *VR_z=NULL;*/
double *Vy=NULL;
double *Vz=NULL;

double OMEGA;   /*OMEGA =omega*2PI*/
double Vin[total_nr_cells_z];

/*double mean_U_A_outflow;
double sum_U_A_outflow;*/

/*****************************************************************************************************************************
DEFINE_EXECUTE_AFTER_CASE UDF
*****************************************************************************************************************************/
/*Use a DEFINE_EXECUTE_AFTER_CASE UDF to calculate the global variables total_nr_faces_inflowface and total_nr_faces_outflowface
in order to prevent mallocing, leading to less memory usage*/
DEFINE_EXECUTE_AFTER_CASE(SR1_flat_disk_fluent12_Vr_correct_Phillips_corrected_atan, libudf)
{
 /*Calculation of the inflow velocities of the centroids of the faces of the inflow plane of the blade*/
 domain = Get_Domain(1);
 zone_ID = 30; /*ID_number for the zone of inflow of the blade*/
 disk_inflow = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the inflow domain of the blade*/

 /**Count nr of faces loop**/
 /**This loop runs over all the faces of the inflow plane, to count the total number of faces of the inflow plane of blade**/
 i=0;
 begin_f_loop(f,disk_inflow)
 {
  i=i+1;
 }
 end_f_loop(f,disk_inflow)

 total_nr_faces_inflowface=i;

 Message("Content of variable total_nr_faces_inflowface is: %d\n", total_nr_faces_inflowface);
 /**End Count nr of faces loop**/
 /**End Calculation of the number of faces of the inflow plane of the blade**/


 /**Calculation of the number of faces of the outflow plane of the actuator disk**/
 zone_ID = 28; /*ID_number for the zone of outflow of the actuator disk*/
 disk_outflow = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the outflow domain of the actuator disk*/
```

```
 /**Count nr of faces loop**/
 /**This loop runs over all the faces of the outflow plane, to count the total nr of faces of the outflow plane of the disk**/
 i=0;
 begin_f_loop(f,disk_outflow)
 {
  i=i+1;
 }
 end_f_loop(f,disk_outflow)

 total_nr_faces_outflowface=i;

 Message("Content of variable total_nr_faces_outflowface is: %d\n", total_nr_faces_outflowface);
 /**End Count nr of faces loop**/
 /**End Calculation of the number of faces of the outflow plane of the actuator disk**/
}
/************************************************************************************************************************
End DEFINE_EXECUTE_AFTER_CASE UDF
************************************************************************************************************************/



/************************************************************************************************************************
 VELOCITY IN X-DIRECTION
************************************************************************************************************************/
DEFINE_PROFILE(axial_velocity, t1, indices)
{
 /*****************************
 Declaration of local variables
 *****************************/
 if(!rs)
 {
  Message("Allocating rs\n");
  rs = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

if(!V_axial_flatdisk)
 {
  Message("Allocating V_axial_flatdisk\n");
  V_axial_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

  if(!U_A_flatdisk)
 {
  Message("Allocating U_A_flatdisk\n");
  U_A_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 if(!Vin_flatdisk)
 {
  Message("Allocating Vin_flatdisk\n");
  Vin_flatdisk = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 /**Define the import variables from matlab files**/
 /*Polynomial coefficients C1 & C2 describing the local pathlines C1*x+C2*/
 /*float C_1[total_nr_cells_z];
 float C_2[total_nr_cells_z];
 FILE *inputFileC2;
 FILE *inputFileC1;
 /**End Define the import variables from matlab files**/

 /**Define variables belonging to the inflow_face of the blade**/
 double r_inflow_centroid[total_nr_faces_inflowface]; /*Radius of the centroid of the faces of inflow plane of actuator*/
 double V_axial_centroid_in[total_nr_faces_inflowface]; /*Velocity of the centroid of the faces of inflow plane of actuator*/
 int rnew_in[total_nr_faces_inflowface];
 double checkpoint_in;
 int newcheckpoint_in;
 int positie_in;
 int teller_in;

 double inflow_centroid_r[total_nr_faces_inflowface];
 double inflow_centroid_V[total_nr_faces_inflowface];
 double old_inflow_centroid_r[total_nr_faces_inflowface];
 double old_inflow_centroid_V[total_nr_faces_inflowface];
 int size_inflow_array;
 double max_inflow_centroid_r;
 double min_inflow_centroid_r;
 double arranged_inflow_centroid_r[total_nr_faces_inflowface];
 double arranged_inflow_centroid_V[total_nr_faces_inflowface];
 int position_min;
 double old_positionvector[total_nr_faces_inflowface];
 int maximum_position;
 int maximum_pos[total_nr_faces_inflowface];
 int max_pos;
 /**End Define variables belonging to the inflow_face of the blade**/

 /**Define variables belonging to the flatdisk**/
 double r_outflow_centroid[total_nr_faces_outflowface];
 double saved_outflow_centroid_r[total_nr_faces_outflowface];
```

```
 /*double xcoord[total_nr_faces_outflowface];
 double ycoord[total_nr_faces_outflowface];
 double zcoord[total_nr_faces_outflowface];*/

 double dr_end;

 int r_outnew[total_nr_faces_outflowface]; /*Integer to ensure the needed significant digits*/
 double checkpoint_out;
 int newcheckpoint_out;
 int positie_out;
 int teller_out;
 double outflow_centroid_r[total_nr_faces_outflowface];
 double old_outflow_centroid_r[total_nr_faces_outflowface];
 int size_outflow_array;
 double max_outflow_centroid_r;
 double min_outflow_centroid_r;
 double arranged_outflow_centroid_r[total_nr_faces_outflowface];
 int position_min_out;
 double old_positionvector_out[total_nr_faces_outflowface];
 int maximum_position_out;
 int maximum_pos_out[total_nr_faces_outflowface];

 /*double r_begin[total_nr_cells_z];*/

 /*double V_i;*/
 /*double Vxs;*/
 /*double omega_s;*/
 double drs;
 /*double rs[total_nr_cells_z];*/
 double Rs;
 double difference_p[total_nr_cells_z];
 double dT_centroid[total_nr_cells_z];
 double check_thrust1;

 double VT_inf[total_nr_cells_z];
 double U_A_outflow[total_nr_cells_z];
 double V_axial[total_nr_cells_z];

 int r_new[total_nr_cells_z]; /*Integer to ensure the needed significant digits*/
 int newcheckpoint[total_nr_faces_outflowface];
 int positie;

 /*FILE *file1;
 FILE *file2;
 FILE *file3;
 FILE *file4;
 FILE *file5;
 FILE *file6;
 FILE *file7;*/
 /*FILE *file8;
 FILE *file9;
 /*FILE *file10;
 FILE *file11;*/
/* FILE *file12;
 /**End Define variables belonging to the flatdisk**/
 /*********************************
 End Declaration of local variables
 *********************************/

 /**Import variables from matlab files**/
/* inputFileC1 = fopen("pathline_C1.txt", "rb");
 i=0;
 while(!feof(inputFileC1))
 {
  fscanf(inputFileC1, "%f", &C_1[i]);
  i++;
 }
 fclose(inputFileC1);


 inputFileC2 = fopen("pathline_C2.txt", "rb");
 i=0;
 while(!feof(inputFileC2))
 {
  fscanf(inputFileC2, "%f", &C_2[i]);
  i++;
 }
 fclose(inputFileC2);
 /**End Import variables from matlab files**/

 /**Calculate r_inflow_centroid and the inflow velocities of the centroids, V_axial_centroid_in.**/
 /*file1 = fopen("check1.dat","w");*/
 j=0;
 begin_f_loop(f, disk_inflow)
 {
  F_CENTROID(x,f,disk_inflow);   /*Calculates the faces' centroid*/
  r_inflow_centroid[j]=sqrt((x[1])*(x[1])+(x[2])*(x[2]));  /*Radius computation of centroid at faces of inflow plane
  of actuator*/
  V_axial_centroid_in[j]= VINF;  /*Assume a uniform inflow velocity*/
  /*fprintf(file1,"%.6g \t %.6g \n",r_inflow_centroid[j], V_axial_centroid_in[j]);*/
```

```
 j=j+1;
 }
 end_f_loop(f, disk_inflow)
/* fclose(file1);*/
 /**End Calculate r_inflow_centroid and the inflow velocities of the centroids, V_axial_centroid_in.**/

 /**It is sufficient to define the r_inflow_centroid until 4 digits after the comma. In order to isolate the same
 r_inflow_centroid from the total array, define an int which is (1000* r_inflow_centroid). Do the same thing for
 the reference values checkpoint and call thz corresponding
 integers newcheckpoint*/
 /**Define new integer rnew to ensure the needed significant digits**/
 for (j=0; j<=total_nr_faces_inflowface-1; j++)
 {
  rnew_in[j] =pow(10,3)*r_inflow_centroid[j];
 }
 /**Define new integer rnew to ensure the needed significant digits**/
 /**Loop to extract the same r_inflow_centroid from the array and declaration of the corresponding inflow velocities**/
 for (j=0; j<=total_nr_faces_inflowface-1; j++)
 {
  for (n=j+1; n<=total_nr_faces_inflowface-1; n++)
  {
   checkpoint_in = r_inflow_centroid[j];
   newcheckpoint_in=pow(10,3)*checkpoint_in;
   if (newcheckpoint_in == rnew_in[n])
   {
    if(newcheckpoint_in!=0)
    {
   positie_in = n;
     r_inflow_centroid[positie_in]= 0;
   V_axial_centroid_in[positie_in] = 0;
    }
   }
  }
 }
 /**Loop to extract the same r_inflow_centroid from the array and declaration of the corresponding inflow velocities**/

 /**Count how many r_inflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/
 teller_in=0;
 for (j=0; j<=total_nr_faces_inflowface-1; j++)
 {
  if (r_inflow_centroid[j]!=0)
  {
   teller_in=teller_in+1;
  }
 }

 Message("teller_face is: %d\n", teller_in);
 /*for (j=0; j<total_nr_faces_inflowface-1; j++)
 {
 Message("Content of variable  r_inflow_centroid is: %f\n",  r_inflow_centroid[j]);
 }
 /**End Count how many r_inflow_centroid are non-zero.**/


 /**Gather all non-zero elements of r_inflow_centroid in one single array called new_r_inflow_centroid**/
 /**Gather all non-zero elements of V_axial_centroid_in in one single array called V_in_centroid**/
  /*file2 = fopen("check2.dat","w");*/
  i=0;
  for (j=0; j<=total_nr_faces_inflowface-1; j++)
  {
   if (r_inflow_centroid[j]!=0)
   {
    inflow_centroid_r[i] = r_inflow_centroid[j];
  old_inflow_centroid_r[i] = inflow_centroid_r[i];
  inflow_centroid_V[i] = V_axial_centroid_in[j];
  old_inflow_centroid_V[i] = inflow_centroid_V[i];
  /*fprintf(file2,"%.6g \t %.6g \t %.6g \t %.6g \n",old_inflow_centroid_r[i], inflow_centroid_r[i],
  inflow_centroid_V[i], old_inflow_centroid_V[i]);*/
  i=i+1;
   }
  }
  size_inflow_array = i;
  /*fclose(file2);*/
  /*grootte_double = sizeof(double);*/

  Message("size_inflow_array is: %d\n", size_inflow_array);
  /*for (j=0; j<=teller-1; j++)
  {
   Message("Content of inflow_centroid_r is: %f\n",  inflow_centroid_r[j]);
  }*/
 /**End Gather all non-zero elements of r_inflow_centroid in one single array called new_r_inflow_centroid**/
 /**End Gather all non-zero elements of V_axial_centroid_in in one single array called V_in_centroid**/
 /**Arrange the values of r_inflow_centroid from small to large and find the corresponding Velocity**/
 /*file3 = fopen("check3.dat","w");*/
 for (j= 0; j<=size_inflow_array-1; j++)
 {
  i=1;
  /**Find maximum value of array inflow_centroid_r**/
  max_inflow_centroid_r = 0;
  for (n=0; n<=size_inflow_array-1; n++)
```

```
 {
  if (inflow_centroid_r[n]> max_inflow_centroid_r)
  {
   max_inflow_centroid_r = inflow_centroid_r[n];
 maximum_position = n;
  }
 }
 maximum_pos[j] = maximum_position;
 /**End Find maximum value of array inflow_centroid_r**/
 /**Find minimum value of array r_random**/
 min_inflow_centroid_r = max_inflow_centroid_r;
 for (n=0; n<=size_inflow_array-1; n++)
 {
  if (inflow_centroid_r[n]< min_inflow_centroid_r)
  {
   min_inflow_centroid_r = inflow_centroid_r[n];
   position_min = n;
  }
 }
 /**End Find minimum value of array r_random**/
  old_positionvector[j]= position_min;
  arranged_inflow_centroid_V[j] = old_inflow_centroid_V[position_min];

 /**Arrange inflow_centroid_r from small to large**/
 arranged_inflow_centroid_r[j] = min_inflow_centroid_r;
 /**Replace inflow_centroid_r[position_min] with max_inflow_centroid_r so that we can find the next minimum**/
 inflow_centroid_r[position_min]= max_inflow_centroid_r;
 /*fprintf(file3,"%.6g \t %.6g \t %.6g \t %.6g \n",old_inflow_centroid_r[j], arranged_inflow_centroid_r[j],
 old_inflow_centroid_V[j], arranged_inflow_centroid_V[j]);*/
}
/*fclose(file3);*/
 /**The last entry of arranged_inflow_centroid_r will be the max of old_inflow_centroid_r. So the last entry
 of arranged_inflow_centroid_V must be the velocity that corresponds to the max of old_inflow_centroid_r, which position is given
 by the first entry of maximum_pos, which stores the position values of the maxima**/
 max_pos = maximum_pos[0];
 arranged_inflow_centroid_V[teller-1] = old_inflow_centroid_V[max_pos];
 Message("Content of last arranged_inflow_centroid_V is: %f\n",  arranged_inflow_centroid_V[teller-1]);
 /*Check of the answers*/
 /*for (j=0; j<=size_inflow_array-1; j++)
 {
  Message("Content of  arranged_inflow_centroid_r is: %f\n",  arranged_inflow_centroid_r[j]);
  Message("Content of  arranged_inflow_centroid_V is: %f\n",  arranged_inflow_centroid_V[j]);
 }
 for (j=0; j<=size_inflow_array-1; j++)
 {
  Message("Content of old_inflow_centroid_r is: %f\n",  old_inflow_centroid_r[j]);
  Message("Content of old_inflow_centroid_V is: %f\n",  old_inflow_centroid_V[j]);
 }
 /*End Check of the answers*/
/**End Arrange the values of r_inflow_centroid from small to large and find the corresponding Velocity**/

 /**Calculate the radii of the centroids of flatdisk**/
 zone_ID = 14; /*ID_number for the zone of the flatdisk*/
 flatdisk = Lookup_Thread(domain, zone_ID); /*Moves the pointer to the flatdisk domain*/
 dr_end =(RP-RTE)/total_nr_cells_z;  /*Calculation of the elemental width dr_end*/
 OMEGA = 2*PI*omega;
 /*file4 = fopen("check4.dat","w");*/
 i=0;
 begin_c_loop(c, flatdisk)
 {
  C_CENTROID(x,c,flatdisk);   /*Calculates the cell's centroid*/
  r_outflow_centroid[i]=sqrt((x[1])*(x[1])+(x[2])*(x[2]));  /*Radius computation of centroid of the cells of the flatdisk*/
  saved_outflow_centroid_r[i] = r_outflow_centroid[i];       /*We need this variable to now the V_axial_flatdisk for each
  cell within flatdisk*/
  /*xcoord[i] = x[0];
  ycoord[i] = x[1];
  zcoord[i] = x[2];*/
  /*fprintf(file4," %.6g \n",r_outflow_centroid[i]);*/
  i=i+1;
 }
 end_c_loop(c, flatdisk)
 /*fclose(file4);*/
 /**End Calculate the radii of the centroids of flatdisk**/

 /**It is sufficient to define the r_outflow_centroid until 4 digits after the comma. In order to isolate the same
 r_outflow_centroid from the total array, define an int which is (1000* r_outflow_centroid). Do the same thing for
 the reference values checkpoint and call the corresponding
  integers newcheckpoint*/
 /**Define new integer rnew to ensure the needed significant digits**/
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  r_outnew[j] =pow(10,3)*r_outflow_centroid[j];
 }
 /**End Define new integer r_outnew to ensure the needed significant digits**/
 /**Loop to extract the same r_outflow_centroid from the array**/
 for (j=0; j<=total_nr_faces_outflowface-1; j++)
 {
  for (n=j+1; n<=total_nr_faces_outflowface-1; n++)
  {
   checkpoint_out = r_outflow_centroid[j];
```

```
  newcheckpoint_out=pow(10,3)*checkpoint_out;
  if (newcheckpoint_out == r_outnew[n])
  {
   if(newcheckpoint_out!=0)
   {
positie_out = n;
    r_outflow_centroid[positie_out]= 0;
   }
  }
 }
}
/**Loop to extract the same r_outflow_centroid from the array and declaration of the corresponding outflow velocities**/

/**Count how many r_outflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/
teller_out=0;
for (j=0; j<=total_nr_faces_outflowface-1; j++)
{
 if (r_outflow_centroid[j]!=0)
 {
  teller_out=teller_out+1;
 }
}
Message("teller_out is: %d\n", teller_out);
/**End Count how many r_outflow_centroid are non-zero. This must be equal tot the total_nr_cells_z**/

/**Gather all non-zero elements of r_outflow_centroid in one single array called new_r_outflow_centroid**/
/*file5= fopen("check5.dat","w");*/
i=0;
for (j=0; j<=total_nr_faces_outflowface-1; j++)
{
 if (r_outflow_centroid[j]!=0)
 {
  outflow_centroid_r[i] = r_outflow_centroid[j];
old_outflow_centroid_r[i] = outflow_centroid_r[i];
/*fprintf(file5,"%.6g \t %.6g \n",outflow_centroid_r[i], old_outflow_centroid_r[i]);*/
i=i+1;
 }
}
size_outflow_array = i;
/*fclose(file5);*/

Message("size_outflow_array is: %d\n", size_outflow_array);
/*for (j=0; j<=teller_out-1; j++)
{
 Message("Content of outflow_centroid_r is: %f\n",  outflow_centroid_r[j]);
}*/
/**End Gather all non-zero elements of r_outflow_centroid in one single array called new_r_outflow_centroid**/

/**Arrange the values of r_outflow_centroid from small to large**/
/*file6= fopen("check6.dat","w");*/
for (j= 0; j<=size_outflow_array-1; j++)
{
 i=1;
 /**Find maximum value of array inflow_centroid_r**/
 max_outflow_centroid_r = 0;
 for (n=0; n<=size_outflow_array-1; n++)
 {
  if (outflow_centroid_r[n]> max_outflow_centroid_r)
  {
   max_outflow_centroid_r = outflow_centroid_r[n];
 maximum_position_out = n;
  }
 }
 maximum_pos_out[j] = maximum_position_out;
 /**End Find maximum value of array outflow_centroid_r**/
 /**Find minimum value of array r_random**/
 min_outflow_centroid_r = max_outflow_centroid_r;
 for (n=0; n<=size_outflow_array-1; n++)
 {
  if (outflow_centroid_r[n]< min_outflow_centroid_r)
  {
   min_outflow_centroid_r = outflow_centroid_r[n];
   position_min_out = n;
  }
 }
 /**End Find minimum value of array r_random**/
 old_positionvector_out[j]= position_min_out;

 /**Arrange outflow_centroid_r from small to large**/
 arranged_outflow_centroid_r[j] = min_outflow_centroid_r;
 /**Replace outflow_centroid_r[position_min] with max_outflow_centroid_r so that we can find the next minimum**/
 outflow_centroid_r[position_min_out]= max_outflow_centroid_r;
 /*fprintf(file6,"%.6g \t %.6g \n", old_outflow_centroid_r[j], arranged_outflow_centroid_r[j]);*/
}
/*fclose(file6);*/
/*Check of the answers*/
/*for (j=0; j<=size_outflow_array-1; j++)
{
 Message("Content of  arranged_outflow_centroid_r is: %f\n",  arranged_outflow_centroid_r[j]);
}
```

```
  for (j=0; j<=size_outflow_array-1; j++)
  {
   Message("Content of old_outflow_centroid_r is: %f\n",  old_outflow_centroid_r[j]);
  }*/
  /*End Check of the answers*/
  /**End Arrange the values of r_outflow_centroid from small to large and find the corresponding Velocity**/

  /**Calculate the r_begin corresponding to arranged_outflow_centroid_r**/
  /*file7= fopen("check7.dat","w");*/
  for (j=0; j<=total_nr_cells_z-1; j++)
  {
   /*r_begin[j] = C_1[j]*x_begin+C_2[j];*/
   Vin[j] = VINF;
   /*fprintf(file7,"%.6g \t %.6g \t %.6g \n", r_begin[j], C_1[j], C_2[j]);*/
  }
  /*fclose(file7);*/
  /**End Calculate the r_begin corresponding to arranged_outflow_centroid_r**/


   /**Now that we have total_nr_cells_z non-zero arranged_outflow_centroid_r arranged from small to large and
   corresponding Vin, one can determine
   the elemental thrust for these elements.*/
   /*filedT = fopen("dT.dat","a+b");*/
  /* file8= fopen("check8.dat","w");*/
   for (i=0; i<=total_nr_cells_z-1; i++)
   {
    V_i =sqrt((pow(VINF,2)/4)+((pow(OMEGA,2)*pow(RP,2))/4)*
    (1-sqrt(1-((4*T)/(PI*(pow(RP,2)-pow(RTE,2))*RHO*pow(OMEGA,2)*pow(RP,2))))))-(VINF/2);
    Vxs = VINF+2*V_i;
    omega_s = (4*(VINF+2*V_i)*V_i)/(OMEGA*pow(RP,2));
    drs =sqrt((VINF+V_i)/Vxs)*dr_end;  /*Calculation of the elemental width drs*/
    rs[i] =sqrt(((VINF+V_i)/Vxs)*(pow(arranged_outflow_centroid_r[i],2)));
    Rs =sqrt((VINF+V_i)/Vxs)*RP;
    difference_p[i] = -(RHO*pow(omega_s,2)/2)*(pow(Rs,2)-pow(rs[i],2));
    dT_centroid[i] =(Vxs-VINF)*RHO*Vxs*((2*PI)*rs[i]*drs)+difference_p[i]*((2*PI)*rs[i]*drs); /*Elemental thrust of
    an annular element*/
    /*Check values. Make sure that the root is taken of a positive number*/
    /*sqrt_Vi= 1-((4*T)/(PI*(pow(RP,2)-pow(RTE,2))*RHO*pow(OMEGA,2)*pow(RP,2)));*/
    /*fprintf(filedT,"%.6g \t %.6g \n", arranged_outflow_centroid_r[i], dT_centroid[i]);*/
    /*fprintf(file8,"%.6g \t %.6g \n", arranged_outflow_centroid_r[i], dT_centroid[i]);*/
   }
   /*fclose(file8);*/
   /*close(filedT);*/
   /*End Loop to calculate the elemental thrust of the centroids of the outflow plane (dT_centroid)*/

   /*Check whether the thrust calculation is correct. The total thrust is divided over all the individual faces
   of the outflow plane.Since we have total_nr_cells_z different outflow_centroid_r, we will have total_nr_cells_z
   elemental thrusts. The total thrust is equal to the sum of all the elemental thrusts of each face of the outflow plane.
   So the sum of all dT_centroid*(total_nr_faces_outflowface/total_nr_cells_z) must equal the total Thrust T*/
   check_thrust1 = dT_centroid[0];

   for (i=0; i<=total_nr_cells_z-2; i++)
   {
   check_thrust1 = check_thrust1 + dT_centroid[i+1];
   check_thrust1 = check_thrust1;
   }

   /*Check whether the thrust calculation is correct. All dT_centroids must be positive*/
   j=0;
   for (i=0; i<=total_nr_cells_z-1; i++)
   {
    if (dT_centroid[i]>=0)
{
 j=j+1;
}
   }
   Message("Content of check_thrust1 is: %f\n", check_thrust1);
  Message("Content of j is: %d\n", j );
  /*End Check whether the thrust calculation is correct. The sum of all dT_centroid must equal T*/

  /**Calculation of the axial velocity at flatdisk**/
   /*file9= fopen("check9.dat","w");*/
   for(i=0; i<=total_nr_cells_z-1; i++)
  {
   /*VT_inf[i] = omega_s*arranged_outflow_centroid_r[i];
   U_A_outflow[i] = 0.5*(-Vin[i] + sqrt(pow(Vin[i],2) +
   VT_inf[i] +dT_centroid[i]/(RHO*PI*arranged_outflow_centroid_r[i]*dr_end)));*/
   V_axial[i]= Vin[i] +V_i;

   /*fprintf(file9,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \n", arranged_outflow_centroid_r[i], dT_centroid[i], VT_inf[i],
   U_A_outflow[i], V_axial[i]);*/
  }
   /*sum_U_A_outflow = U_A_outflow[0];

   for (i=0; i<=total_nr_cells_z-2; i++)
   {
   sum_U_A_outflow = sum_U_A_outflow + U_A_outflow[i+1];
   sum_U_A_outflow = sum_U_A_outflow;
   }*/
```

```
  /*fclose(file9);*/
  /**End Calculation of the axial velocity at flatdisk**/

  /**Find V_axial_flatdisk for total_nr_faces_outflowface number of cells**/
  /**The axial velocity is now known for total_nr_cells_z, which is axisymmetric so now we have to impose the right V_axial
  to the right saved_outflow_centroid_r. the array saved_outflow_centroid_r must be used since the original r_outflow_centroid
   is changed because of the sorting prog**/
  /**It is sufficient to define the arranged_r_outflow_centroid until 4 digits after the comma. In order to isolatethe same
   saved_outflow_centroid_r from the total array, define an int which is (1000* arranged_r_outflow_centroid).
   Do the same thing for the reference values checkpoint
   and call the corresponding integers newcheckpoint*/
  /**Define new integer rnew to ensure the needed significant digits**/
  /*file10= fopen("check10.dat","w");*/
  for (j=0; j<=total_nr_cells_z-1; j++)
  {
   r_new[j] =pow(10,3)*arranged_outflow_centroid_r[j];
   /*fprintf(file10," %.6g \n",arranged_outflow_centroid_r[j]);*/
  }
  /*fclose(file10);*/

   /*file11= fopen("check11.dat","w");*/
   for (n=0; n<=total_nr_faces_outflowface-1; n++)
  {
   newcheckpoint[n] =pow(10,3)*saved_outflow_centroid_r[n];
   /*fprintf(file11," %.6g \n", saved_outflow_centroid_r[n]); */
  }
  /*fclose(file11);*/
  /**End Define new integer r_outnew to ensure the needed significant digits**/
  /**Loop to extract the same r_outflow_centroid from the array**/

  for (j=0; j<=total_nr_cells_z-1; j++)
  {
   for (n=0; n<=total_nr_faces_outflowface-1; n++)
   {
    if (r_new[j] == newcheckpoint[n])
    {
     if(newcheckpoint[n]!=0)
     {
      positie = n;
      V_axial_flatdisk[positie]= V_axial[j];
      Vin_flatdisk[positie]= Vin[j];
      U_A_flatdisk[positie] = V_i;
     }
    }
   }
  }

  /*file12= fopen("check12.dat","w");
  for (j=0; j<=total_nr_faces_outflowface-1; j++)
  {
  fprintf(file12,"%.6g \t %.6g \n", V_axial_flatdisk[j], Vin_flatdisk[j]);
  }
  fclose(file12);*/

  /**End Find V_axial_flatdisk for total_nr_faces_outflowface number of cells**/
  /**Impose the axial velocity using F_PROFILE**/
  i=0;
  begin_c_loop(c, t1)
  {
   F_PROFILE(c, t1, indices) = V_axial_flatdisk[i];
   i=i+1;
  }
  end_c_loop(c, t1)
  /**End Impose the axial velocity using F_PROFILE**/

}
/*****************************************************************************************************************************
End VELOCITY IN X-DIRECTION
*****************************************************************************************************************************/


/*****************************************************************************************************************************
 VELOCITY IN Y-DIRECTION
*****************************************************************************************************************************/
/*This part calculates the y-velocity inside the actuator disk volume. The y-velocity consists of the y-component of the
tangential velocity and the y-component of the radial velocity.
The tangential velocity is calculated first. Folowed by the computation of the radial velocity*/

DEFINE_PROFILE(y_velocity, t1, indices)
{
 if(!VT_z)
 {
  Message("Allocating VT_z\n");
  VT_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
 }

 if(!Vy)
 {
  Message("Allocating Vy\n");
  Vy = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
```

```
}

 /*if(!VR_z)
{
 Message("Allocating VR_z\n");
 VR_z = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
}*/

double VT_y[total_nr_faces_outflowface];
/*double VR_y[total_nr_faces_outflowface];*/

/*******************************
Declaration of local variables
*******************************/
double r_centroid_flatdisk;
double ycoord;
double zcoord;
double VT;
/*double VR;*/
int z_pos;
double theta;

/*FILE *fileben;*/
/*********************************
End Declaration of local variables
*********************************/
/**Calculate first the tangential velocity inside the flatdisk**/
i=0;
/*fileben = fopen("check_Vtan.dat","w");*/
begin_c_loop(c, t1)
{
 C_CENTROID(x,c, t1);   /*Calculates the cells' centroid*/
 r_centroid_flatdisk = sqrt((x[1])*(x[1])+(x[2])*(x[2]));
 ycoord = x[1];
 zcoord = x[2];
 /*VT = (4*PI*U_A_flatdisk[i]*(Vin_flatdisk[i]+U_A_flatdisk[i]))/(2*PI*r_centroid_flatdisk*OMEGA);*/
 VT = ((VINF+V_i)/Vxs)*omega_s*r_centroid_flatdisk;
 /**End Calculate first the tangential velocity inside the flatdisk**/

 /**Calculation of radial velocity**/
 /*mean_U_A_outflow =  mean_U_A_outflow/total_nr_cells_z;
 VR =  (mean_U_A_outflow*PI*r_centroid_flatdisk)/(4*RP);*/
 /**End Calculation of radial velocity**/

  /**Calculation of the angle Theta**/
  if (ycoord!=0)
  {
   theta = fabs(atan(zcoord/ycoord));
  }
  else
  {
   theta = PI/2;
  }
  /**End Calculation of the angle Theta**/

  /**General Condition for VT_y**/
  if (zcoord<0)
  {
   VT_y[i] = pow(-1, sense+1)*VT*sin(theta);
   /*VR_z[i] = VR*sin(theta);*/
  }
  if (zcoord>0)
  {
   VT_y[i] = pow(-1, sense)*VT*sin(theta);
   /*VR_z[i] = -VR*sin(theta);*/
  }
  /**End General Condition for VT_y**/

  /**General Condition for VT_z**/
  if (ycoord<0)
  {
   VT_z[i] = pow(-1, sense)*VT*cos(theta);
   /*VR_y[i] = VR*cos(theta);*/
  }

  if (ycoord>0) /*z<0 & y!=0*/
  {
   VT_z[i] = pow(-1, sense+1)*VT*cos(theta);
   /*VR_y[i] = -VR*cos(theta);*/
  }
  /**End General Condition for VT_z**/
  /**End Calculation of VT_y and VT_z**/

  /*Vy[i] = VT_y[i]+VR_y[i];*/
  Vy[i] = VT_y[i];
  F_PROFILE(c, t1, indices) = Vy[i];
 /*fprintf(fileben,"%.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \t %.6g \n",OMEGA,
  ycoord, zcoord, theta, r_centroid_flatdisk, Vin_flatdisk[i], V_axial_flatdisk[i], VT, VT_y[i], VT_z[i]);*/

  i=i+1;
```

```
  }
  end_c_loop(c, t1)
  /*fclose(fileben);*/

  /**End Calculation of Vr_y and Vr_z**/

/*   Message("Content of counter i is: %d\n", i );
   Message("Content of counter teller is: %d\n", teller );
   Message("Content of counter teller2 is: %d\n", teller2 );



  for (j=0; j<=2*total_nr_cells_z-1; j++)
  {
  Message("Content of VT_y[i] is: %f\n", VT_y[j] );
  /*Message("Content of VT_z[i] is: %f\n", VT_z[j] );*/
  /*Message("Content of V_y[i] is: %f\n", V_y[j] );*/
/*  }*/

}

/************************************************************************************************************************
End VELOCITY IN Y-DIRECTION
************************************************************************************************************************/


/************************************************************************************************************************
 VELOCITY IN Z-DIRECTION
************************************************************************************************************************/
DEFINE_PROFILE(z_velocity, t1, indices)
{
  if(!Vz)
  {
   Message("Allocating Vz\n");
   Vz = (double *) malloc(total_nr_faces_outflowface*sizeof(double));
  }


 i=0;
 begin_c_loop(c, t1)
 {
  /*Vz[i]=VT_z[i]+VR_z[i];*/
  Vz[i]=VT_z[i];
  F_PROFILE(c, t1, indices) = Vz[i];
  i=i+1;
 }
  end_c_loop(c, t1)
}
/************************************************************************************************************************
End VELOCITY IN Z-DIRECTION
************************************************************************************************************************/
```

Appendix C

m-file which calculates spanwise lift and drag distribution

```
%Author: Marilyne Lino
%        Master student Aerodynamic design
%        Faculty of Aerospace Engineering
%        University of Technology Delft (TUD)
%This file is created to extract CL and CD data from static pressure and
%Tau_wall data of Fluent in matlab.
%date: 16-04-10
%last updated:16-04-10
close all;
clear all;
clc;

%----definition of variables----%
yref=40.41/2;
p_operating=23840;

%----Import datafiles------%
nr_of_sections=57;
for j=0:nr_of_sections-1

    filename=strcat('section',int2str(j));

    datafile=dlmread(filename, '', 1,1);
%----End Import datafiles------%

%----Sort the datafile ------%
%Sort the datafile such that x is given in ascending order and order the
%other vectors in the same manner. This is done to ensure that we have for
%all rownumbers the corresponding data in all vectors
datasorted=sortrows(datafile,1);
%----End datafile----%

%----define rows of x-,y-&z-coord, pressure, twx & twz----% (differs for
%manual and jou extraction----%
xcoord=1;
ycoord=2;
zcoord=3;
pres=6;
twx=5;
twz=4;

%----Split the airfoil in lower and upper section----%
%To split the airfoil in an upper and lower part find the expression
%(linear relation) which describes chord of the airfoil

%Determination of the expression that describes the chord in terms of the
%linear relation: z=slope*x+b
%Determination of the slope
xmin_chord=datasorted(1,xcoord);
xmax_chord=datasorted(length(datasorted),xcoord);
zmin_chord=datasorted(1,zcoord);
zmax_chord=datasorted(length(datasorted),zcoord);
```

```
chord_slope=(zmax_chord-zmin_chord)/(xmax_chord-xmin_chord);
chord_constant=zmin_chord-chord_slope*xmin_chord;
koorde(j+1)=xmax_chord-xmin_chord;
%End Determination of the expression that describes the chord

j_up=1;
j_bot=1;

for i=1:length(datasorted)
    if (datasorted(i,zcoord)-(chord_slope*datasorted(i,xcoord)+chord_constant)>=0)
          upperdata(j_up,:)=datasorted(i,:);
          j_up=j_up+1;
    else
        lowerdata(j_bot,:)=datasorted(i,:);
        j_bot=j_bot+1;
    end
end

% plot(datasorted(:,xcoord), datasorted(:,zcoord), 'ro');
% hold on
% plot(upperdata(:,xcoord), upperdata(:,zcoord), 'b+');
% plot(lowerdata(:,xcoord), lowerdata(:,zcoord), 'sk');
% figure(2)
% plot(datasorted(:,xcoord), datasorted(:,zcoord), 'r-');
%----End Split the airfoil in lower and upper section----%

%----Calculate x and z component of Ps----%
%Upper part
for i=1:length(upperdata)-1
    Ps_mean_upper(i)=((upperdata(i,pres)+upperdata(i+1,pres))/2)+p_operating;
    l_upper(i)=sqrt((upperdata(i+1,zcoord)-upperdata(i,zcoord))^2+(upperdata(i+1,xcoord)-upperdata(i,xcoord))^2);
    Ftx_upper(i)=((upperdata(i,twx)+upperdata(i+1,twx))/2)*l_upper(i);
    Ftz_upper(i)=((upperdata(i,twz)+upperdata(i+1,twz))/2)*l_upper(i);

    if (upperdata(i+1,xcoord)-upperdata(i,xcoord)~=0)
        slope_upper(i)=abs(atan((upperdata(i+1,zcoord)-upperdata(i,zcoord))/(upperdata(i+1,xcoord)-upperdata(i,xcoord))));
    else
        slope_upper(i)=0;
    end

    if (upperdata(i+1,zcoord)>=upperdata(i,zcoord) && slope_upper(i)~=0)
        Fpsx_upper(i)=Ps_mean_upper(i)*l_upper(i)*sin(slope_upper(i));
        Fpsz_upper(i)=-Ps_mean_upper(i)*l_upper(i)*cos(slope_upper(i));
    elseif (upperdata(i+1,zcoord)<upperdata(i,zcoord) && slope_upper(i)~=0)
        Fpsx_upper(i)=-Ps_mean_upper(i)*l_upper(i)*sin(slope_upper(i));
        Fpsz_upper(i)=-Ps_mean_upper(i)*l_upper(i)*cos(slope_upper(i));

    end
end

    Ps_mean_upper(length(upperdata))=((upperdata(1,pres)+lowerdata(1,pres))/2)+p_operating;
    l_upper(length(upperdata))=sqrt((upperdata(1,zcoord)-lowerdata(1,zcoord))^2+(upperdata(1,xcoord)-lowerdata(1,xcoord))^2);
    Ftx_upper(length(upperdata))=((upperdata(1,twx)+lowerdata(1,twx))/2)*l_upper(length(upperdata));
    Ftz_upper(length(upperdata))=((upperdata(1,twz)+lowerdata(1,twz))/2)*l_upper(length(upperdata));
    slope_upper(length(upperdata))=abs(atan((upperdata(1,zcoord)-lowerdata(1,zcoord))/(upperdata(1,xcoord)-lowerdata(1,xcoord))));
    Fpsx_upper(length(upperdata))=Ps_mean_upper(length(upperdata))*l_upper(length(upperdata))*sin(slope_upper(length(upperdata)));
    Fpsz_upper(length(upperdata))=Ps_mean_upper(length(upperdata))*l_upper(length(upperdata))*cos(slope_upper(length(upperdata)));

    Ps_mean_upper(length(upperdata)+1)=((upperdata(length(upperdata),pres)+lowerdata(length(lowerdata),pres))/2)+p_operating;
    l_upper(length(upperdata)+1)=sqrt((upperdata(length(upperdata),zcoord)-lowerdata(length(lowerdata),zcoord))^2+...
    ...(upperdata(length(upperdata),xcoord)-lowerdata(length(lowerdata),xcoord))^2);
    Ftx_upper(length(upperdata)+1)=((upperdata(length(upperdata),twx)+lowerdata(length(lowerdata),twx))/2)*l_upper(length(upperdata)+1);
    Ftz_upper(length(upperdata)+1)=((upperdata(length(upperdata),twz)+lowerdata(length(lowerdata),twz))/2)*l_upper(length(upperdata)+1);
    slope_upper(length(upperdata)+1)=abs(atan((upperdata(length(upperdata),zcoord)-..
    ...lowerdata(length(lowerdata),zcoord))/(upperdata(length(upperdata),xcoord)-lowerdata(length(lowerdata),xcoord))));
    Fpsx_upper(length(upperdata)+1)=-Ps_mean_upper(length(upperdata)+1)*l_upper(length(upperdata)+1)*sin(slope_upper(length(upperdata)+1));
    Fpsz_upper(length(upperdata)+1)=-Ps_mean_upper(length(upperdata)+1)*l_upper(length(upperdata)+1)*cos(slope_upper(length(upperdata)+1));

%Lower part
for i=1:length(lowerdata)-1
    Ps_mean_lower(i)=((lowerdata(i,pres)+lowerdata(i+1,pres))/2)+p_operating;
    l_lower(i)=sqrt((lowerdata(i+1,zcoord)-lowerdata(i,zcoord))^2+(lowerdata(i+1,xcoord)-lowerdata(i,xcoord))^2);
    Ftx_lower(i)=((lowerdata(i,twx)+lowerdata(i+1,twx))/2)*l_lower(i);
    Ftz_lower(i)=((lowerdata(i,twz)+lowerdata(i+1,twz))/2)*l_lower(i);

    if (lowerdata(i+1,xcoord)-lowerdata(i,xcoord)~=0)
        slope_lower(i)=abs(atan((lowerdata(i+1,zcoord)-lowerdata(i,zcoord))/(lowerdata(i+1,xcoord)-lowerdata(i,xcoord))));
    else
        slope_lower(i)=0;
    end

    if (lowerdata(i+1,zcoord)>=lowerdata(i,zcoord) && slope_lower(i)~=0)
        Fpsx_lower(i)=-Ps_mean_lower(i)*l_lower(i)*sin(slope_lower(i));
        Fpsz_lower(i)=Ps_mean_lower(i)*l_lower(i)*cos(slope_lower(i));
    elseif (lowerdata(i+1,zcoord)<lowerdata(i,zcoord) && slope_lower(i)~=0)
        Fpsx_lower(i)=Ps_mean_lower(i)*l_lower(i)*sin(slope_lower(i));
        Fpsz_lower(i)=Ps_mean_lower(i)*l_lower(i)*cos(slope_lower(i));
    end
end
```

```
%Calculation lift and drag caused by Ps for lower and upper part seperately
L_Ps_upper=sum(Fpsz_upper);
L_Ps_lower=sum(Fpsz_lower);
D_Ps_upper=sum(Fpsx_upper);
D_Ps_lower=sum(Fpsx_lower);

%Calculation lift and drag caused by t(tau) for lower and upper part seperately
L_t_upper=sum(Ftz_upper);
L_t_lower=sum(Ftz_lower);
D_t_upper=sum(Ftx_upper);
D_t_lower=sum(Ftx_lower);

%Calculation of lift and drag caused by Ps and t
L_Ps_tot=L_Ps_upper+L_Ps_lower;
D_Ps_tot=D_Ps_upper+D_Ps_lower;
L_t_tot=L_t_upper+L_t_lower;
D_t_tot=D_t_upper+D_t_lower;

%Calculation of total lift and drag
L_tot=L_Ps_tot+L_t_tot;
D_par_tot=D_Ps_tot+D_t_tot;

%Calculation of lift and drag coefficients
rho=0.3796;
Vinf=177.91;
dynamic_press=0.5*rho*Vinf^2;
S=75.29214;
c_ref=S/yref;
A=(yref^2)/S;
e=1.78*(1-0.045*A^0.68)-0.64;
t = datasorted(1,ycoord);

if (t< 14.7) && (9.69<=t && t<=10.722) || (14.69<=t && t<=15.72)
    c(j+1)=0.156*t+2.34;
    c_ypos(j+1)=datasorted(1,ycoord);
else
    c(j+1)=koorde(j+1);
    c_ypos(j+1)=datasorted(1,ycoord);
end



CL_Ps=L_Ps_tot/(dynamic_press*c(j+1));
CL_t=L_t_tot/(dynamic_press*c(j+1));
CD_Ps=D_Ps_tot/(dynamic_press*c(j+1));
CD_t=D_t_tot/(dynamic_press*c(j+1));
CL_tot=L_tot/(dynamic_press*c(j+1));
CD_par_tot=D_par_tot/(dynamic_press*c(j+1));
CDi=CL_tot^2/(pi*A*e);
CD_tot=CD_par_tot+CDi;


%----Determine the y-coordinate to know span position----%
span_position=(yref-datasorted(1,ycoord))/yref;
%----End Determine the y-coordinate to know span position----%

spanwise_Data(j+1,:)=[span_position CL_Ps CL_t CL_tot CD_Ps CD_t CD_par_tot CDi CD_tot];

clear datafile datasorted upperdata lowerdata i j ;
clear -regexp F Ps slope lower upper coord chord CL CD _tot tw;

end

%Save spanwise_Data in a mat file
savefile='Datafile_spanwise_distribution.mat';
save(savefile, 'spanwise_Data');

%plot spanwise liftdistribution
figure(3)
plot(spanwise_Data(:,1),spanwise_Data(:,4), '-ok');
xlabel('spanwise position [-]');
ylabel('CL [-]');
title('CL distribution');

%plot spanwise dragdistribution
figure(4)
plot(spanwise_Data(:,1),spanwise_Data(:,9),'-ok');
xlabel('spanwise position [-]');
ylabel('CD [-]');
title('CD distribution');


%Save data in file to plot with the other results
savefile2='Datafile_spanwise_distribution_inboardup.mat';
save(savefile2, 'spanwise_Data');

figure(5)
plot(spanwise_Data(:,1),spanwise_Data(:,7),'-ob');
hold on;
```

```
plot(spanwise_Data(:,1),spanwise_Data(:,8),'-+r');
xlabel('spanwise position [-]');
ylabel('CD [-]');
title('CD distribution');
legend('CD_{par}', 'CD_i');
```

Appendix D

Journal files which extract airfoils along the wing span and their data

D.1 Journal file which extracts airfoils along the wing span

```
;;
surface/iso-surface/y-coordinate
section0
le_inbetween_engines
le_outboard
le_root
le_tip
lower_wing_inbetween_engines
lower_wing_midsection
lower_wing_outboard
lower_wing_root
lower_wing_tip
te_midsection
te_root
te_tip
upper_wing_midsection
upper_wing_root
upper_wing_tip
exhaust_inboard_engine
exhaust_outboard_engine
hub_nose_inboard_engine
hub_nose_outboard_engine
intake_inboard_engine
intake_outboard_engine
midpart_nacelle_inboard_engine
mid_part_nacelle_outboard_engine
nacelle_inboard_engine_wing_intersection
nacelle_outboard_engine_wing_intersection
oil_cooling_inboard_engine
oil_cooling_outboard_engine
underneath_prop_inboard_engine
underneath_prop_inboard_engine:83
underneath_prop_inboard_engine:84
underneath_prop_inboard_engine:85
underneath_prop_outboard_engine
underneath_prop_outboard_engine:86
```

```
underneath_prop_outboard_engine:87
underneath_prop_outboard_engine:88


0.05


quit
;;

etc.
```

D.2 Journal file which extracts airfoil data

```
;;
file/export/ascii
section0
section0

no
pressure
x-wall-shear
z-wall-shear
quit
no
quit
quit
;;

etc.
```