INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Aero-Structural Blade Design of a High-Power Wind Turbine

Bruno Miguel Santo António Tojo

Dissertação para a obtenção de Grau de Mestre em

Engenharia Aeroespacial

Júri

Presidente:	Prof. Fernando José Parracho Lau
Orientador:	Dr. André Calado Marta
Vogal:	Prof. Aurélio Lima Araújo

Julho 2012

Acknowledgments

I would like to begin by thanking my supervisor, Dr. André Marta, for all the fruitful conversations we had along this work and also for his advice on the direction of the project and for his extraordinary help on the writing of this document.

I also would like to thank Prof. Fernando Lau and all the other teachers from the Aerospace Engineering course at IST. Without them passing their knowledge, it would not be possible to reach this stage. I also thank Prof. Aurélio Araújo for taking the time to review and evaluate my work.

Finally, I would also like to thank my family, because it would not be possible to be here without their support, specially my parents, who keep believing in me even when I do not believe myself. I also want to thank all my friends and colleagues for all their help during my studies and specially to Angela, who is always there for me and helps me more than it is possible to ask someone.

Resumo

Este trabalho centra-se na criação de um modelo aero-estrutural para análise de um rotor de uma turbina eólica de elevada potência de eixo horizontal. Ao longo do trabalho procedeu-se a caracterização detalhada da pá, ao desenvolvimento de um modelo de interacção fluido-estrutura adequado e, finalmente, à junção de ambos com o recurso um adequado processamento dos resultados.

A parametrização da pá baseou-se no conceito do disco actuador, o qual foi melhorado por simulações de mecânica dos fluidos computacional, e o modelo da estrutura foi definido como uma longarina, a qual tem a forma da pá. Para abordar o fenómeno de interacção fluido-estrutura, desenvolveu-se um programa para OpenFoam dedicado a simulação de rotores de turbinas eólicas. O acoplamento fluido-estrutura recorreu a uma estratégia de acoplamento simples, na qual a análise da estrutura e a análise do fluido são feitos independentemente, combinados apenas pela actualização das condições de fronteira. Em relação ao movimento de rotação das pás, este foi definido de acordo com um sistema de referencial não-inercial único, isto é, todo o domínio definido está em rotação conjunta com o rotor da turbina a velocidade angular constante. O domínio estrutural foi simulado de forma linear elástica, possibilitando a modelação de pequenas deformações resultantes das forcas aerodinâmicas distribuídas.

Os resultados obtidos com a simulação são coerentes com o expectável ao nível das velocidades do fluido e das distribuições de pressão. Atendendo a que se espera que o tamanho dos rotores de turbinas eólicas de eixo horizontal continue a aumentar, os resultados aqui apresentados confirmam que os deslocamentos causados pelas forças induzidas pela passagem do fluido são um problema importante a considerar no desenho de novas pás.

Embora não seja ainda possível analisar as consequências desses efeitos na potência retirada da turbina, apresenta-se um método de processamento que possibilita essa avaliação. Para que se obte nham resultados relevantes é necessário continuar o desenvolvimento do modelo aqui apresentado, nomeadamente introduzindo as propriedades correctas na modelação de pás de turbinas eólicas.

Palavras-chave: Turbina Eólica, Interação Fluído-Estrutura, Mecânica dos Fluídos Computacional, Análise de Alta-Fidelidade, Referencial Não-Inercial Único, OpenFOAM, ICEMCFD

Abstract

The present work focused on the development of an aero-structural design framework for a high-power horizontal axis wind turbine. To achieve this, it was necessary to carefully characterize the blade, to develop a suitable fluid-structure interaction solver and, finally, to combine both with post-processing tools.

The blade parameterization was inspired in the disc actuator concept, which was then improved through computational fluid dynamics simulations and, the structure was modeled as main spar that has the shape of the blade. It was developed a fluid-structure interaction solver for OpenFOAM, dedicated to simulate wind turbine rotors. The fluid-structure coupling was achieved through a loose coupling strategy, which means that there are separated solvers for the flow and structure analysis, which are combined through the update of boundary conditions. To simulate the blades rotation movement, it was used an approach based on the single rotating frame method, meaning that the whole domain rotated with the turbine rotor with a constant angular velocity. For the structural domain, a linear elastic solver was used, able to model the small deformations due to the distributed aerodynamic forces.

The simulations of the rotor produced valid and interesting results namely, correct flow fields and pressure distributions. Considering that it is expected horizontal axis wind turbine rotors to continue growing to even larger sizes than the one modeled, it was shown that blade displacements due to flow induced forces are definitely a problem that needs to be taken into account when designing new wind turbine blades.

Although it was not possible to analyze the consequences of these effects in the turbine output power, a post-processing method to make this evaluation was presented. To obtain relevant results on this subject, it would be important to further develop the structural solver and model, giving them the correct proprieties to model wind turbine blades.

Keywords: Wind Turbine, Fluid-Structure Interaction, CFD Development, High-Fidelity Analysis, Single Rotating Frame, OpenFOAM, ICEMCFD

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, there will be a brief introduction to the content of this thesis. Beginning with the motivation for this project, followed by the proposed goals. It will also be presented a small introduction to the physics behind the wind energy conversion and the current and future renewable energy panorama.

1.1 Motivation

In today's world, the demand for power is increasing everyday and, as a consequence, so is the research for new and better ways to produce electricity. It can be produced in many different ways but, until now, none can be as efficient, powerful and clean as wind energy conversion. Having this in mind, it becomes even more important to find new ways to improve wind turbines so that in the future, wind energy can continue to push the industry towards a cleaner and more sustainable direction.

This can only become a reality if wind energy can become economically competitive. As the source of this type of energy is already free, the only possibility to improve the competitiveness of wind energy is by improving the efficiency of aerogenerators. As stated by Casas et al. [1], the efficiency of the blade is determinant for the wind turbine economical performance. Therefore, any improvement in the aerodynamic and structural design of the the blades implies a significant profit increase during the 20 years in which the turbine is expected to operate.

The present work is focused on simulating and optimizing a blade for a high power Horizontal Axis Wind Turbine(HAWT). These turbines are now becoming more popular because they can be placed offshore, taking advantage of the more frequent and powerful winds and the fewer size restrictions in these areas, thus producing more energy than was ever possible inshore and, eliminating some of the disadvantages of these wind farms such as noise pollution and landscape changes [2].

1.2 Goals and Objectives

- Elaborate a small literature review on wind energy turbines, fluid and structural models and optimization techniques;

- Parameterize the geometry of a HAWT blade, in order to obtain good fluid and structural computation meshes;
- Evaluate and develop numerical solvers to simulate the flow around a HAWT blade, the structural analysis and fluid-structure interaction;
- Perform aerostructural simulations of HAWT blades;
- Discuss the possibilities for the optimization process in the blade.

1.3 Wind Energy

Even though the first references of using wind energy report to the Babylonian empire in the seventeenth century B.C. [3], this subject remains today as current and revolutionary as it has always been.

The main idea is quite simple, the air, as any moving object, has kinetic energy that can be used to power the movement of a rotor. The theoretical power available in the wind is given by

$$P = \frac{1}{2}\rho_a A_T U^3, \tag{1.1}$$

where A_T is the cross-sectional area of the rotor, ρ_a is the air density and U is the wind speed.

However, a turbine cannot extract all the power from the wind stream passing through, as only a part of this kinetic energy is transfered to the blade. So, it is necessary to define a power coefficient (C_P) as the ratio between the actual power developed and the theoretical power available in the wind. This is then the first part where the engineer can intervene, as the actual power produced depends on many engineering factors, such as the profile of the rotor blades, blade arrangements and settings, etc.([3])

According to the actuator disc concept [4],in which the flow around a wind turbine is represented by a two-dimensional flow similar to a diffuser and the turbine is simplified as a disc (Fig.1.1), it is considered that the disc induces a velocity variation on the free stream velocity (U_∞), the velocity on the disc (U_d) is defined by

$$U_d = U_\infty(1 - a), \tag{1.2}$$

where a is called the flow induction factor.

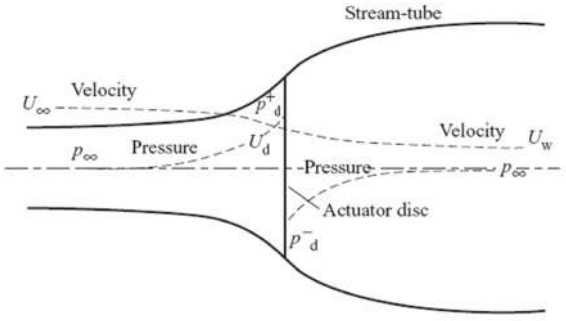


Figure 1.1: Energy extracting actuator disc and stream-tube schematic. In [4].

Applying the Bernoulli's equation to calculate the pressure differential caused by the turbine and defining the power produced by the turbine as the work done by the force caused by this differential, it is possible to get C_P as a function of the flow induction factor,

$$C_P = 4a(1-a)^2. \tag{1.3}$$

This definition brings an interesting conclusion that optimal value for a is $1/3$ and the maximum value possible for C_P is 0.593, which is known as the Betz limit [5].

Just like an airplane wing, wind turbine blades work by generating lift due to their shape, the differential of pressure between the sides of the blade, pushes it resulting in a lift force perpendicular to the direction of flow of the air. As in any lifting surface, it would be impossible to deflect a flow without production drag. Since drag is in the downwind direction, it may seem that it would not matter for a wind turbine as drag would be parallel to the turbine axis, so would not slow the rotor down. When the rotor is stationary, this is indeed the case. However the blade's own movement through the air means that, as far as the blade is concerned, the wind is blowing from a different angle. This effect is more visible further from the center of rotation, as the tangential velocity increases with radius, the flow that reaches the blade is then called apparent wind. The apparent wind is stronger than the true wind but its angle is less favorable: it rotates the angles of lift and drag to reduce the effect of lift force pulling the blade round and increase the effect of drag slowing it down. It also means that the lift force contributes to the "thrust" on the rotor and not only torque, as is explicit in Fig.1.2

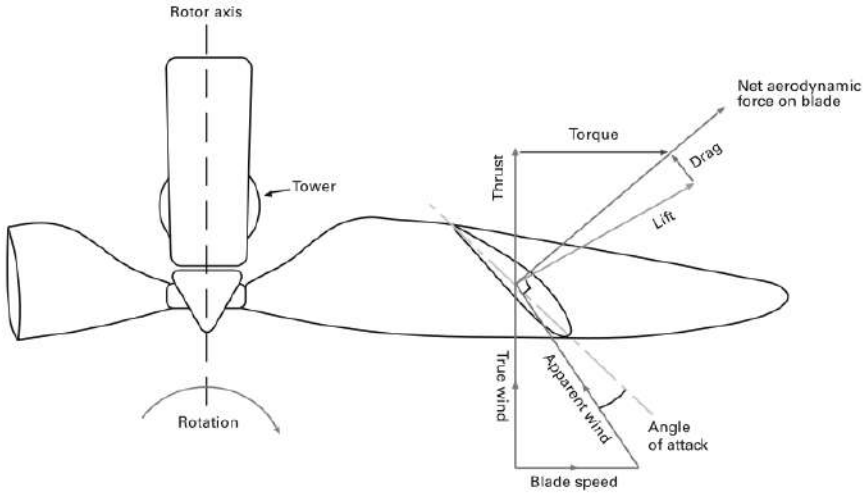


Figure 1.2: Schematic of forces and air speeds in wind energy extraction. In [4].

Mainly over the last 30 years, engineering studies in aerodynamics, structural dynamics, and micrometeorology have contributed to a 5% annual increase in the energy yield of wind turbines [6]. As an example in 1992, the blades designed for $40m$ diameter rotors could produce $500kW$, while current designs can go up to $100m$ in diameter and produce about $3MW$ of power as the one presented in [7], having achieved a relative reduction in the blade weight [8], while continually improving the C_P can reach up to 0.5, which is already very close the theoretical maximum. These improvements were accomplished by designs that push the design margins for blade materials and structures. To achieve a reliable design

within these reduced design margins, a new degree of sophistication had to be introduced into the design process, which will be discussed in the next chapter.

Wind energy extraction is such an important topic that almost all the aspects of its process are being studied all over the world. In the USA, the National Renewable Energy Laboratory (NREL), which funds and regulates all the American research on this subject, is also responsible for developing benchmark wind turbines, which are often used to validate results or as comparison for new models and ideas. In [9], it is stated that the USA wind electricity generation will have a major boost in the next few year, increasing from the current 1.5% up to 20% of the electricity needed in 2030.

In Europe, all of European Union (EU) countries have agreed to a major increase in the amount of the energy consumed in the countries that will have to be generated by wind power by 2020, as shown in Fig.1.4. This increase in power will be achieved, not only by installing more wind power farms but also by improving the efficiency of the extraction through research. This is coordinated by European Wind Energy Association (EWEA) and mainly made in national laboratories as the Danish Risø, as well as in Universities, as in which Delft University has a lead. Most of the implementation is carried out by private companies such as Vestas and GE Energy, which are the clear industry leaders. All these major players are now focusing their efforts to push offshore wind energy extraction and agree that this is the future of wind energy [10–12] .

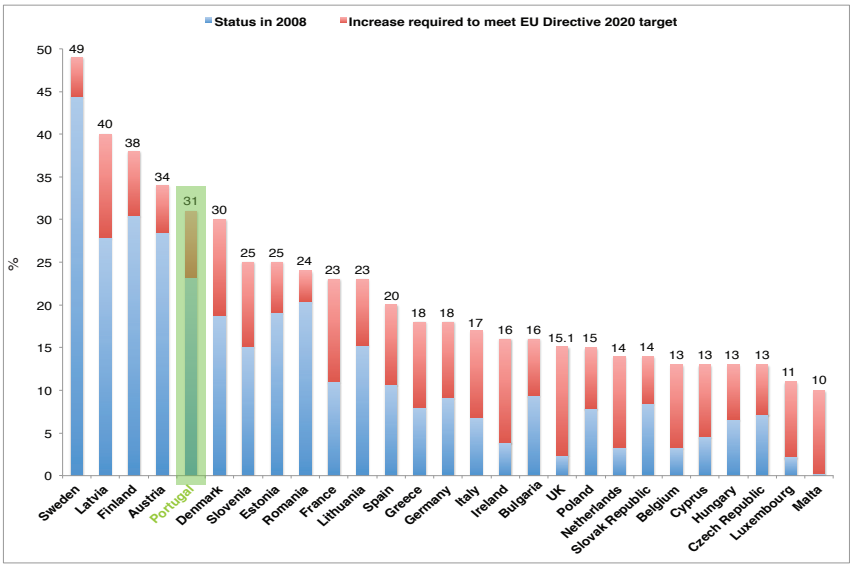


Figure 1.3: Renewable Energy Share. From data in [10].

Portugal, which has now around 23% of the energy consumption generated by renewable energy sources (Fig.1.3), is one of the top EU countries in this sector, and it is believed that will reach its renewable energy action plan goal, maintaining its spot on the Top 5 of the renewable energy producers in EU. Although, in absolute values, it is impossible to be in the lead due to the dimensions of the country and the big investments from the largest EU economies which were trailing behind in 2010 as UK and France, which aspire to become leaders in 2020. Fig. 1.4 shows that in 2020 Portugal will be in pair with countries that are considered to have stronger economies and leaders in innovation as Italy, Netherlands, Denmark and Sweden, and that in 2010 was producing much more wind energy from the

wind than those countries.

According to the EWEA, Portugal will be rising its wind power capacity to $6.9GW$, producing $14.6TWh$, covering almost 23% of electricity consumption and resulting in 31% of energy exclusively from renewable sources by 2020. Furthermore, EWEA points out that wind production capacity factors may have been underestimated in the national plan, so the final figure could be even higher than the forecasts. The country had $3.9GW$ already installed in 2010, and in 2011 the installed power kept growing despite the European economical crisis to reach $4.08GW$ by the end of the year [13].

New wind projects are now being allocated and investments made into innovation and development of offshore wind technology, as the first offshore wind turbine in Portugal was installed recently, in May 2012, for experimental tests. By 2015, the first offshore power plants will be installed, which should have a capacity of $25MW$, the remaining $50MW$ offshore will be installed until 2020 (Fig.1.5). In this area, it was decided to constrain the investment in research so that, when investment in hardware is made, it is possible to install much more mature technology, becoming a smarter investment.

APREN (Portuguese renewable energy association) notes that the national plans forecasts have not taken into account the economic crisis or savings resulting from energy efficiency measures. Increasing an additional $3GW$ of wind power by 2020 will depend on several factors such as electricity demand, ability to transfer consumption from peak to off-peak hours, cost of offshore wind, market share growth of electric vehicles and environmental impact. In APRENs view, Portugal will exceed its goals only if it is capable of exporting the surplus electricity from renewable sources. The peripheric position of Portugal and the existing limitations in power grid interconnection, especially between Spain and France, currently presents the major obstacle to this.

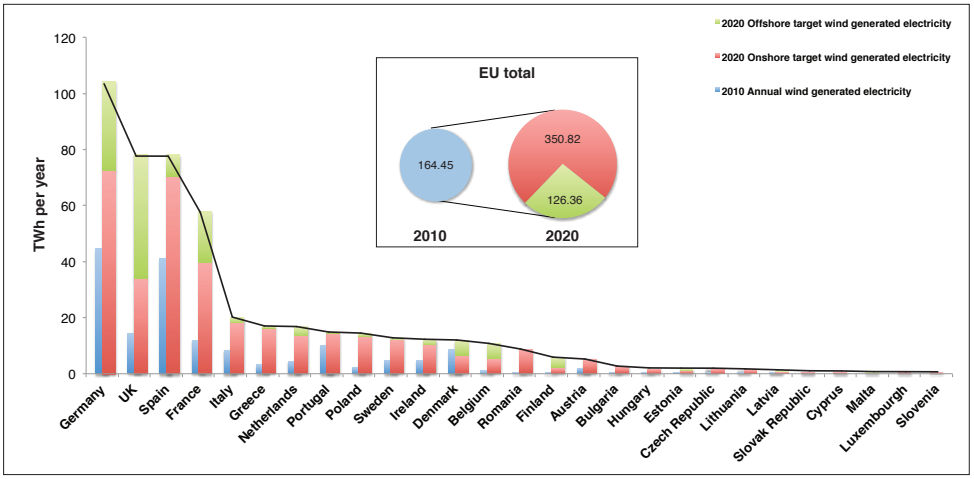


Figure 1.4: Targets to wind power electricity for 2020. From data in [10].

1.4 Overview

After this brief introduction, chapter 2 will consist on a more extensive literature review focusing the on the relecant topics for this work, including computational fluid dynamics and finite element analysis of wind turbines, fluid structure interaction projects and optimization techniques.

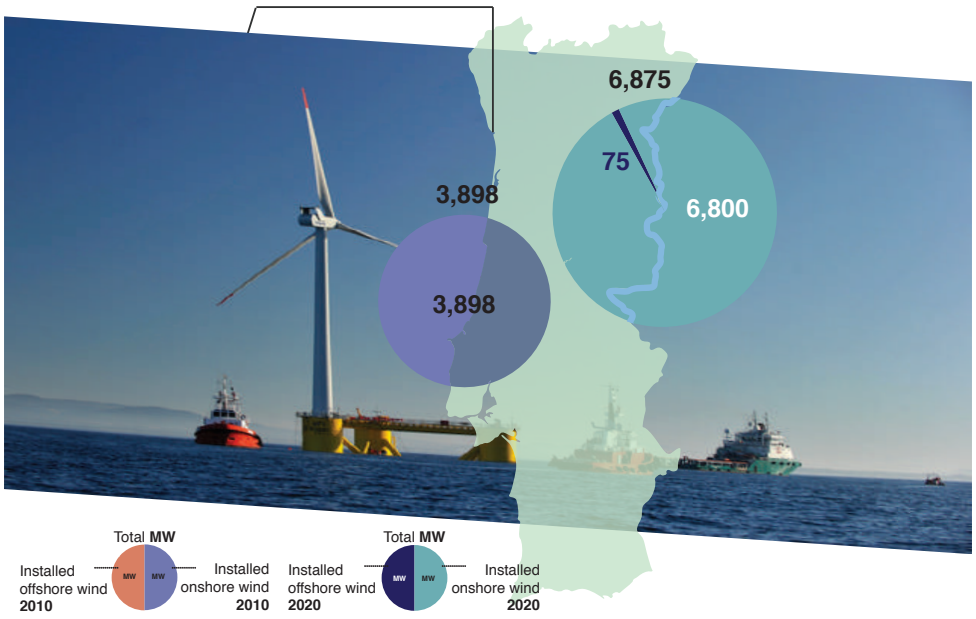


Figure 1.5: Installed wind energy power capacity in Portugal, and installation of the experimental offshore HAWT. From data in [10] and [13].

Chapter 3 will present the methodology used in this work to generate the wind turbine blades meshes. From the parameterization of the blade to the choice of the appropriate mesh generator. This chapter will also explain the code used to generate the mesh and the tools developed for post processing the simulations results.

In chapter 4 it is was explored OpenFoam toolbox, including the definition of the main solvers and boundary conditions to be used. In this chapter, it is also presented the development of the fluid-structure interaction solvers which were created specifically for this project and which probably represent the best contribution from this work to the advance of aero-strutural research in wind turbines.

Following the solvers presentation, chapter 5 will be focused on the simulations and results obtain during the course of this work, including a detailed explanation of the different steps and choices made to reach the final simulation of the wind turbine rotor.

Chapter 6 will then present the main discussions and conclusions obtained.

Chapter 2

Literature Review

Starting to address the objectives of this work, it will be presented a small review of the state of the art of scientific research in aerodynamic and structural models for wind turbine blades, finishing with coupled models and optimization technique .

2.1 Aerodynamic and Structural Models

2.1.1 Aerodynamics and Computational Fluid Dynamics (CFD)

The effectiveness of a wind turbine farm is largely based on the capacity of rotor blades to extract energy from the incoming wind under given reliability, sustainability and safety standards [14]. The aerodynamics of the blade are so important for generating wind energy that several airfoils dedicated and optimized for wind turbine blades were developed. Delft University (DU) was one of the major contributers in this area. As stated in [15], the characteristics of a wind turbine airfoil are always a trade-off between good aerodynamic and structural characteristics but, for the Delft airfoils, the wish to keep sensitivity to contamination and contour imperfections were the primary design requirement. This was done in order to keep the loss of lift due to contamination to a minimum, although this had the disadvantage of making the maximum lift capacity quite moderate. Comparing to the NACA-airfoils in which leading edge contamination can be traced to the high upper surface velocities and resulting high adverse pressure gradients due to the larger upper surface thickness, giving premature transition and early separation, DU thick airfoils currently in use by most blade manufacturers all have a constrain in upper surface thickness to avoid this premature turbulent separation. To compensate for the resulting loss in lift of the upper surface, a certain amount of lower surface aft loading is incorporated, giving the typical S-shape of the pressure side.

With increasing rotor diameters, blade designers tend to use thick airfoil sections in a large part of the blade. Airfoils with a thickness of 25% are already located at mid-span sections, and thickness increases towards the root, ending in airfoils of about 40% relative thickness. Thick airfoils provide more structural stiffness and enable the designer to reduce weight, giving a reduction of fatigue loads and costs, but this comes with a cost, increased pressure gradients over the aft part of the airfoil upper surface that,

in conjunction with leading edge contamination, may lead to early turbulent separation and a severe reduction of the maximum lift coefficient.

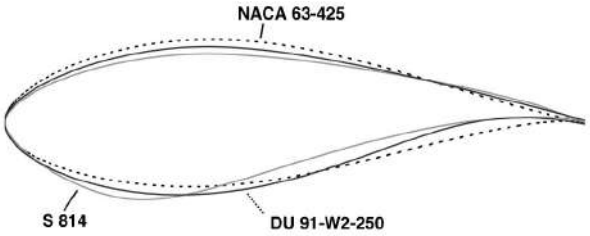


Figure 2.1: Shape of three airfoils with approximately 25% thickness. In [16].

Comparing the performance for the DU, FFA, S8xx, Risø, and NACA airfoil series, which all are commonly used in wind turbines, for both clean and transition-fixed configurations, the measurements indicate that 25% thick airfoils, as the ones presented in Fig. 2.1, have the best overall performance and lowest roughness sensitivity. The impact of vortex generators on the airfoil performance is also examined in [16] and is considered favorable, in particular on stall regulated turbines, as the vortex generator on the airfoil reduces roughness sensitivity, thus improving the performance.

Contamination of the blade leading edge cannot be avoided and field measurements have demonstrated large power reductions due to that. Both rated power and maximum power coefficient are reduced by leading-edge contamination. The actual loss in power coefficient depends on the design of the rotor blade, i.e., the chords at the mid-span and inboard segment, the airfoil choice and the criteria with respect to the optimal tip speed ratio. Severe nose contamination at mid-span can lead to a calculated loss in power coefficient for each blade segment of approximately 4%.

Blade methods of various levels of complexity are used to calculate the aerodynamic loads on a wind turbine. Limiting to consider methods capable to provide realistic global performance predictions at reduced computational effort, the current trend is still to give preference to Blade Element Momentum (BEM) based approaches, whereas more sophisticated Reynolds-averaged Navier-Stokes (RANS) or other viscous-flow codes are used for two-dimensional (2D) simulations or to focus on local analysis of rotating blade flow. The BEM Method is the most currently used due to its low computational cost, this method was introduced by Glauert [17], as combination of one dimension momentum theory and blade element considerations to determine the loads locally along the wing span. BEM assumes that all sections along the rotor are independent and can be treated separately [18], three-dimensional effects(3D) from the tip vortices are taken into account by applying Prandtl's tip loss correction and, after this correction, the flow around a given blade section is assumed to be 2D. So, the success of this method depends on how reliable is the airfoil data for the different blade sections.

Greco el al.[14] presented a performance prediction method that can hardly be grouped into any of the two classes above. The theoretical and computational methodology proposed is based on a general formulation to describe the unsteady, three-dimensional flow around bodies of arbitrary shape immersed in a non-uniform inflow. The mathematical model is based on the assumptions of inviscid, incompressible flow, and hence viscosity effects are neglected and require suitable modeling. Theoretical models of this type are widely used to study fluid-structure interaction (FSI) of rotary-wings for aircraft as well

as seacraft applications. The aim of that work was to extend an existing BEM to study wind turbine performance. Such methodology allows to describe the inviscid flow around three-dimensional bodies of arbitrary shape moving in arbitrary motion with respect to an incoming flow. For wind turbines, deeply affected by viscous effects, a novel formulation capable to accounting for such effects has been presented. A distinguishing feature of the proposed approach is that no semi-empirical relationships are required and the resulting implementation is straightforward. Numerical applications of the proposed methodology show that the estimation of turbine performances is reliable up to stall onset. The introduction of a trailing wake alignment technique has shown to even improve the code reliability for that wind speed range. For separated flow operating conditions, discrepancies with respect to experimental data occur. Hence, a more physically consistent way of accounting for viscous effects has to be considered. Similarly, some issues related with the corrected evaluation of lift induced drag at high blade incidence have been identified and were considered as subjects for further investigation.

When working with typical aeroelastic design codes, computations are usually based on the BEM, relying on tables for airfoil lift and drag. But, as very little data exists for high angles of attack and knowledge of 3D effects is limited, these computations become uncertain. At this stage, it is worthy observing that the knowledge of the 3D unsteady aerodynamic blade loads is fundamental to characterize the behavior of the wind turbine with respect to aeroelastic and aeroacoustic applications. In fact, such loads are the forcing terms of the equations of the blade motion that yield the aeromechanical response of the system, and represent an acoustic source forcing the wave equation that governs the pressure disturbance radiated from the blade to the flow-field. Hence, the capability to model the loads distribution on the blade surface, in a physically consistent way, deeply affects the prediction of the performances of the turbine from an aeroacoustoelastic standpoint. In [19], a Computations Fluid Dynamic(CFD) analysis of a HAWT rotor while parked is presented, improving the knowledge about this case. Using Risø own software to make the meshes and calculations, and Detached-Eddy Simulation (DES) and Reynolds Averaged Navier-Stokes (RANS) models. The CFD codes reproduce the measured trends quite well and give very similar results. The deviations observed can be explained by the difference in the applied turbulence models. The comparison of steady and transient RANS results show that, with respect to mean quantities, the gain of using time true computations is very limited for this case. Finally, the DES methodology provides a much more physical representation of the heavily stalled part of the flow over blades at high angles of attack.

CFD studies are always a clever way to introduce, test and design new blade models. Disgraskar [20] presented a project that is somewhat similar to the one presented in the current work, as the objective was also to simulate the flow around a wind turbine, using OpenFOAM [21]. For this, special boundary conditions were developed, such as cyclic grid interfaces, and a multi reference frame solver, in order to model the wind turbine as accurate as possible. It was possible to model the NREL Phase IV and DU wind turbines, being the former modeled quite successfully, and even the wake was calculated with success, but it was not possible to get good results for the later.

Also using the NREL turbine, in [22] it is presented a comparative study between the theory of the actuator disc, the results of a CFD analysis on a high resolution structured mesh, with advanced turbulence

and transition models, and experimental results, discussing the possibility of creating a hybrid code. It was concluded that the CFD simulation provided an excellent match with experimental data in the attached flow regime. However, the CFD and panel code over-predict peak lift and tend to underestimate stalled flow. The results presented support the use of a calibrated actuator disk methodology based on a normal fidelity CFD modeling approach. Implementing actuator momentum correlations obtained from an equivalent CFD analysis offers a potentially economical methodology for simulating complete multiple wind turbines within a specific environment.

Another CFD study of the NREL PHASE IV wind turbine is presented by Sezer-Uzol [23] using a solver that uses a finite volume formulation of the Navier-Stokes equation for 3D, compressible, unsteady or steady-state. Three conditions were studied in which the inviscid results show that the flow is attached for the low velocity cases, with the case where yaw angle is not zero having an asymmetric wake structure, whereas there is a massive separation over the entire blade span in the post-stall case. Although on a much small scale, the mesh presented in this work served as inspiration for the mesh presented in the current report, as the outer mesh is represented by a cylinder and the region of the blades has a much smaller elements then the outside mesh, size restriction had to be applied as the computations presented in [23] were made using NREL and NASA clusters.

The project presented by Nilsson [24] proposed to study OpenFOAM as a tool for turbulent flow in water turbines. The aim was to validate OpenFOAM performance against both commercial CFD codes and experimental data. In that paper they show that the results obtained were as efficient as the ones achieved using the commercial codes, fitting equally well to the experimental data. Based on this, OpenFOAM can confidently be used as a CFD tool to analyze and design turbines.

Sprague et al. [25] focused on the computational simulation of a full wind-turbine farm, along with its interaction with regional weather. This is a multi-scale, multi-physics problem, as it is necessary to account for atmospheric boundary layer, blade scale turbulence and also structural dynamics of the HAWT. This is an important problem as underperformance of HAWT is believed to be particularly caused by inadequate account of atmospheric effects as well as propagations of turbine wakes. The simulations were made by taking advantage of the periodic boundaries in OpenFOAM, and coupling this software with other to produce the fluid-structure interaction required for a good solution. Once again OpenFOAM demonstrates to be a competent CFD tool, in which is possible to perform FSI successfully.

As HAWT are becoming mature technology, considering alternative options for wind energy generation is also becoming relevant. In [26], it is proposed to study a wing that heaving and pitching simultaneously may extract energy from an incoming flow, thus acting like a turbine. This paper starts from this premise to elaborate a CFD study, in which an airfoil oscillates through a flow. It is shown that under 2D laminar flow conditions, a single airfoil can reach efficiencies as high as 35%. It is also shown that leading edge vortex shedding is actually an important mechanism by which a good synchronization between the vertical and the heaving velocity is achieved, thus yielding positive power extraction over most of the periodic cycle. Also, the Portuguese company Omnidea [27] is working on a similar project in order to extract energy from the wind using a wing-like structure that rises due to lift generated and is then pulled back while producing less lift force, causing torque in a generator and producing energy.

2.1.2 Structural Models and Finite Element Method (FEM)

As part of the design process, a wind turbine, must be analyzed for the aerodynamic, gravitational, inertial and operational loads that will experience during its design life. Although structural analysis has always been important in the wind turbine blade design, mainly because of fatigue problems, as the size of blade become bigger (Fig.2.2), so do the concerns with blade structural design [6, 9].

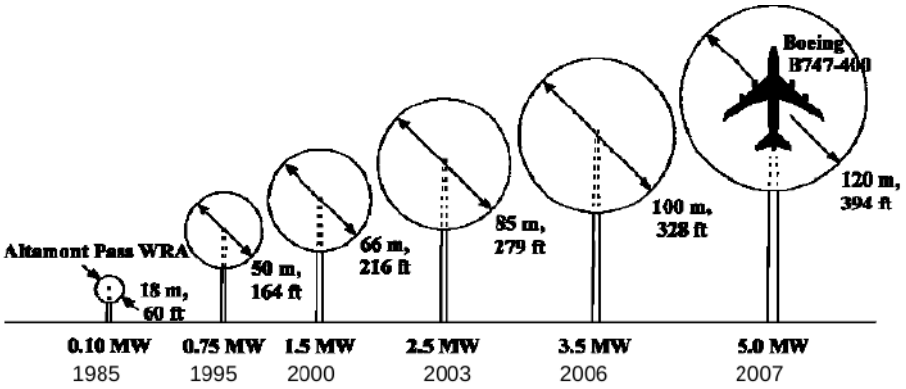


Figure 2.2: Turbine Rotor Growth since 1985 [9].

Jensen et al. [28] described the structural testing and numerical simulation of a 34 m composite HAWT blade with a load carrying main spar. Their work starts with the same premise that the current project starts, that is the fact that HAWT are becoming bigger with a considerable increase in rotor size, which means that structural problems on the blades need to be addressed. The finite element method (FEM) has been used traditionally to investigate global behavior in terms of, for example, eigen frequencies, tip deflections and global stress and strain levels. In their work, the FE model was calibrated with data from a full scale model, which means its accurate and it was possible to evaluate complex loads, modeling actual wind conditions, otherwise impossible on the physical model.

The design of a wind turbine structure involves many considerations such as strength, stability, cost and vibration. In [29], an optimization model for the design of a typical blade structure of HAWT is presented and it is focused on the reduction of vibrations. A good design philosophy for reducing vibration is to separate the natural frequencies of the structure from the harmonics of rotor speed. The main spar was represented by thin-walled tubular beam composed of uniform segments, each of which had different cross-sectional properties and length. The optimization variables were chosen to be the cross-sectional area, radius of gyration and length of each segment. The optimal design is pursued with respect to maximum frequency design criterion. The problem was formulated as a non-linear mathematical programming problem solved by multi-dimensional search techniques. Structural analysis was restricted to the case of uncoupled flapping motion of the rotating blade, where an exact method of solution was given for calculating the natural vibration characteristics. The results presented showed that the approach used was efficient and improved the reference design.

Bechly [30] described the optimization on the structure of a HAWT blade. A computational code was developed to make a detailed finite element mesh from data obtain from blade element theory and panel code predictions. This preliminary blade design was done using NACA 4412 airfoils and varying twist

and chord, similar to the method that will be explained further in this document. The blade was then constructed from fiberglass skin and additional stiffening was added at the hub connection. A detailed finite element mesh of the blade was then built using a propose written program. The finite element results, obtained using a commercial FEM software, compared well with static bending and twisting deflections of the blade and, according to the dynamic analysis, it was even possible to understand that the natural frequencies of the rotating blade are higher than non-rotating, due to stress-stiffening.

Malcolm et al.[31] developed a software to simplify the aero-structural analysis. That software converts a full 3D FEM model of a HAWT blade to an equivalent rectangular beam. This was done by applying a suite of unit tip loads and transfer the displacement results to a series of MATLAB® routines which extract the stiffness matrices for the equivalent beam elements. Then, the stiffness matrices are incorporated into a preprocessor and generate the complete aeroelastic model.

2.1.3 Fluid-Structure Interaction (FSI)

Following the concerns explored in the previous sections, fluid structure interactions(FSI) are currently becoming more relevant in the design of new HAWT. This phenomenon is connected to the deflection of elastic materials when subjected to flow induced loads, as well as the alterations that this deflection produce on the flow, changing its characteristics.

The FSI computations are usually classified in weakly or strongly coupled. For weakly coupled systems, it is common to use partitioned solvers, in which the main problem is how to pass the information between the solvers. On the other hand, for strongly coupled ones, the only choice is to use a single solver that solves both fluid and structural problems, but which is much more complex than the partitioned one [32].

Campbell [33] developed a very similar project to the one presented in this document, but applied to water propellers instead, where the deformation of the blade due to the flow is even more effective. A partitioned FSI solver, using the OpenFOAM software and an author-developed finite element (FE) structural solver, was used to perform FSI simulations of flexible turbomachinery. In addition, it was also developed an inverse FE structural code, using it to compute inverted structural shapes that account for deformations due to fluid loads so that the structures deform into their prescribed design shapes after elastic and viscoelastic deformations occur.

Lorentzon [34] presented an FSI code for OpenFOAM for studying vibrations on a beam. The code developed reproduces the frequency shift well within the margin of error of the mesh quality and the measurement of the time period. The amplitude correlation study, the scaling and the independence of velocity in the frequency increases the confidence that the FSI algorithm worked as intended. Although the method seemed to be stable, an issue appeared while using two sequential solvers, the structural was more accurate and stable than the flow. Therefore, the upper bound for the time step was given by the fluid solver. However, due to the combination between the temporal discretization and the transfer step where viscous traction was determined by displacement field which where the interpolation formula have higher error than the structural solver, a lower bound is obtained from the structural solver. This

implies that the total algorithm seldom achieves the individual solvers optimal performance and there will be situations in which no convergence is obtained, although each individual solver converges.

Aerostructural analysis techniques are a specialization of more general FSI solution methods, where the fluid in question is air, and the structure represents a flexible component such as an aircraft wing or wind turbine blade. Aerostructural design optimization using high-fidelity models is a computationally intensive multidisciplinary design optimization (MDO) problem, usually using high-fidelity aerodynamic analysis in the aerostructural problem while using considerably smaller finite-element models. This is often justified since the primary area of interest in these studies is the aerodynamic performance of the aerostructural system.

Kennedy et al. [35] presented both aerostructural analysis and design sensitivity methods that are designed to be efficient when the aerodynamic and structural disciplines both require significant computational resources and time. The aerodynamic analysis uses a parallel panel code, coupled to a parallel finite-element code for the analysis of composite structures. This finite-element code is specifically designed, including geometric nonlinearity and has analytic design variable sensitivity analysis capability. The inter-disciplinary coupling is handled by passing pressure and displacement values through a parametric surface representation of the outer mold line of an aircraft.

Aerodynamic and structural optimization of HAWT blades has become a subject of considerable interest, as the ultimate goal is to reduce the cost of energy production. This design process is a multidisciplinary task, involving conflicting requirements, such as maximum performance with minimum loads and noise. Moreover, since the turbine operates in a wide range of wind conditions, it can only be optimized for one. In [36], the optimization was achieved by modifying the twist and chord of the blade, as well as the airfoils. A minimum cost of energy was obtained, which is lower then current commercial turbines, but it was also concluded that, with the use of traditional airfoils, it is impossible to reduce the cost of energy from the one that is already achieved.

2.2 Optimization Methods

Most current engineering designs are made using optimization methods, always trying to achieve something lighter, more efficient or simply cheaper.

This technique has a very simple theoretical formulation. Given a set X and a function $f : X \to \Re$ which is the objective function, one wants to find $x^* \in X$ such that, for all $x \in X$, there holds $f(x) \geq f(x^*)$, where x is called the decision variable. X is a subset of $\Re^n$ defined by the constrains in Eq.(2.1), in which I and E are disjoint sets of integers.

$$\{minf(x), x \in \Re^n; c_j(x) \leq 0, j \in I; c_j(x) = 0, j \in E\} \tag{2.1}$$

The way to reach the optimum value for the decision variable is what distinguishes the different optimization methods. There are three conventional methods: calculus-based, enumerative and random methods [37].

Calculus based methods divide into two main classes: indirect and direct. Indirect methods seek local extrema by solving the usually non-linear set of equations resulting from setting the gradient of the objective function equal to zero, while direct methods seek local optima by hopping on the function and moving in a direction related to the local gradient. This is simply the motion of hill-climbing: to find the local best, climb the function in the steepest permissible direction. As major disadvantages, this search method lacks robustness, it is local in scope and depends upon the existence of a derivative, while real world search is fraught with discontinuities and vast multi-modal, noisy search spaces and depicted in a less calculus friendly function.

Enumerative schemes consist on a very straight forward idea, within a finite search space or a discretized infinite search space. The search algorithm starts looking at objective function values at every point in the space, one at a time. Although effective, this is not very efficient because many practical spaces are simply too large to perform an extensive search.

Random search algorithms can not be expected to be much more efficient than enumerative schemes. The genetic algorithm is an example of a search procedure that uses random choice as a tool to guide a highly exploitative search through a coding of parameter space. Using random choices as a tool in a directed search process seems strange at first, but nature contains many examples. Randomized search does not necessarily imply directionless search.

A brief description of gradient-based method, the steepest-descend method, and a heuristic method, the genetic algorithm, is given in the following subsections.

2.2.1 Steepest-Descent Method

The philosophy of the descent methods is to generate a sequence x^k such that each iteration guarantees a decrease of the objective function [37]. Essentially, for all non-optimal x^k ($0 \notin \partial f(x^k)$), there is a direction d^k, which corresponds to the strict separation of the sets $\{0\}$ and $\partial f(x^k)$.

The idea in the steepest-descent method is to find the descend direction d^k, which consists in looking for the best possible descent at each iteration. This is called the steepest-descent direction and is defined by

$$d^k \in Argmin f'(x^k; d). \tag{2.2}$$

Steepest-descend algorithms suffer from important drawbacks which make them inefficient. The computation of the descend direction requires the knowledge of the all the sub-differential domain at each iteration. This is an excessive requirement and, in many applications, it is completely impossible to obtain. Furthermore, nondifferentiable points may cause oscillations in the sequence of $\{x^k\}$ and converge to a non-optimal point.

2.2.2 Genetic Algorithm

Genetic Algorithms (GAs) are search algorithms based on the mechanics of natural selection and natural genetics [38]. They combine survival of the fittest among string structures with a structured yet

randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old one, and an occasional new part is their for good measure.

Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite length string over some finite alphabet. So, the first step of the optimization process is to code the parameter x as a finite length string. In the next step it has to be created a random start with a population of strings, from which there will be generated the successive populations of strings. After this start, successive populations are generated using the genetic algorithm. By working from a population of well-adapted diversity instead of a single point, GAs the probability of finding a false peak is reduced over methods that go point to point.

Many search techniques require much auxiliary information in order to work properly, for example, calculus based techniques need derivatives. In contrast, GAs have no need for this auxiliary information, which is why GAs are said to be blind. As to perform an effective search for better structures they only need the payoff values (the values of the objective function) associated with individual strings, this is crucial for the robustness of the genetic algorithm.

A simple genetic algorithm that yields good results in many practical problems is composed of three operations which are, reproduction, crossover and mutation.

Reproduction is a process in which individual strings are copied according to their objective function values. Copying strings according to their fitness values means that strings with higher value, have higher probability of contributing to one or more offspring in the next generation.

Crossover may proceed in two steps, first all the members of the matting pool are matted randomly. Then, each pair of strings undergoes crossing over which consists of a random trade parameters between the matted strings. This implies that a new generation is created using information from the fittest member of the old one, but none individual passes unchanged between generations.

Mutation is needed because, even though reproduction and crossover effectively search and recombine the best information, occasionally they may become over zealous and lose some potential useful genetic material. In artificial genetic systems the mutation operator protects against such irrecoverable loss.

Even though it is not certain that the individual obtained is the best possible, GAs are an effective way to produce optimized solutions, and this is why it is one of the most used optimization methods.

Mendez et al. [39] used genetic algorithms to optimize the twist and chord distributions in a wind turbine blade. These distributions were computed to maximize the mean expected power, using an approach that avoid assumptions about optimal angle of attack related to the ratio between the lift to drag coefficients. The geometry definition of a wind blade is a problem with many degree of freedom being suitable to fall in local optima, which can be surpassed using evolutionary methods. The results showed that some degree of optimization was possible, but when applied to commercial wind turbines the gain was low. However this author expect that the cost in refinement cyclic to obtain such optimized commercial blades might be reduced using this approach.

Casas [1] presented a genetic optimization algorithm for wind turbines. A simple, fast, and robust

aerodynamic simulator was embedded in the design environment to predict the performance of any turbine produced as intermediate individual of the evolutionary process. This simulator is based on BEM and in order to reduce computations some simplifications were contemplated and the results were corrected by means of the application of neural network based approximations.

Chapter 3

Methodology For Wind Turbine Aero-Structural Analysis

This chapter and the next deal with the methodology for the definition of the model created to simulate the flow around a HAWT blade and its structural counterpart. A considerable amount of time was invested in this part of the work, once it was necessary to start from a very early stage of development as the main research on this subject is being sponsored by private corporations, making most of the data extremely confidential and inaccessible. It was possible to find some simple surface (STL) files from academic research projects but, as it was imperative that the blade was fully parametrized, it was impossible to use such files.

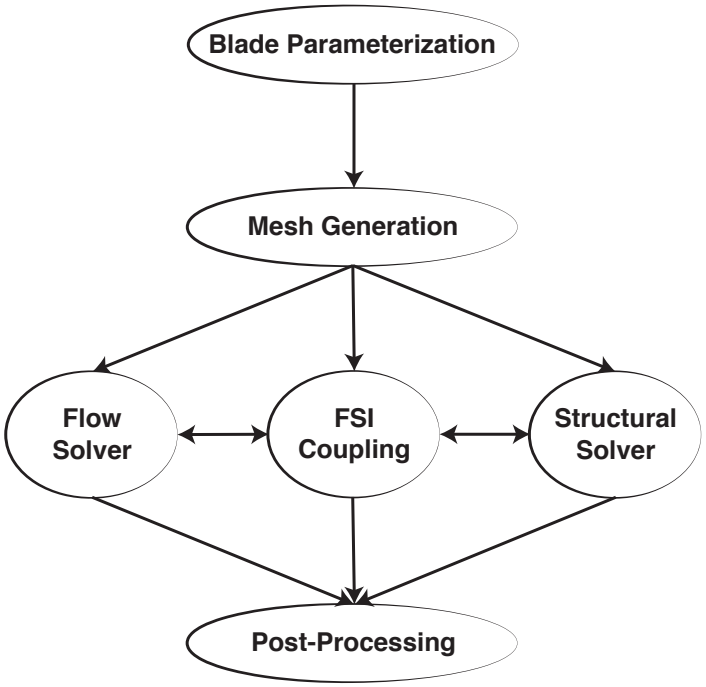


Figure 3.1: Schematic of WT aero-structural blade analysis.

Following the strategy illustrated in Fig.3.1 and, having decided that the toolbox to perform the CFD simulations would be OpenFOAM [21] which, as clarified by the literature review , is a good option for fluid structure interaction simulations, as well as accurate simulations on wind turbines. It was then necessary to evaluate the other parts of the methodology, such as the parametrization of the blade, mesh and geometry generators and finally, post processing of the results once the simulations were performed.

The first important step in a CFD simulation is to decide the dimensionality of the model to use. The simulation should be as simple as possible in order to obtain good results quickly, but still capture the relevant physical phenomena. 2D simulations are often used in the first steps of design and, for many cases, it produces good preliminary results. A full 3D simulation is needed to obtain correct secondary flows. This is specially important in compressors and fans (or HAWT), where the negative pressure gradients make boundary layers grow much quicker than what they do in regular turbines. Having this in mind, it was decided that in a first approach it would be used a simple 2D mesh of a beam that was expected to deform the same way a HAWT blade would. This model was used to test solvers and validate the fluid-structure interaction solver that was developed. Afterward a full 3D model for the HAWT blades it was created, step by step,.

This process had several stages which will be described detailed in the next sections. Before starting with the definition of the meshes, it was necessary to choose a coordinate system. The system chosen is presented in the Fig.3.2, where x points in the direction of the axial flow, which also corresponds to the axis of rotation.

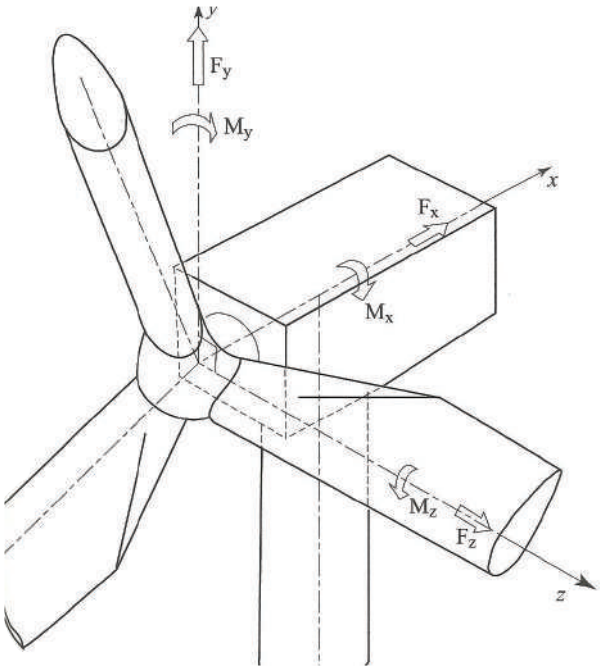


Figure 3.2: Axis of the wind turbine blade.

3.1 Blade Parameterization

As the purpose of wind turbines is to extract as much power from the wind as possible and each component must be optimized for that process.

As a first approach, the surface will be modeled using a discretization presented by Kim et al. [40]. This parametrization is based on the actuator disc concept [4], which is presented in Sec.1.3, and that is why most of the parameters are influenced by the flow induction factor. This discretization starts with the selection of the airfoils, to which a series of coordinate transformations will be applied to obtain a HAWT blade. From the literature reviewed, DU airfoils seemed to be the obvious choice, unfortunately it was impossible to access these airfoils. As such, airfoils from NACA and S8xx series, with similar characteristics, which were easier to access [16].

The power developed by a rotor at a certain wind speed depends on the relative velocity between the wind and the rotor tip. The ratio between these two speeds, called the tip speed ratio(λ), is defined as

$$\lambda = \frac{R\Omega}{U} == \frac{2\pi N_r R}{60U}\,, \tag{3.1}$$

where Ω is the angular velocity, R is the radius of the rotor and N_r is the rotational speed of the rotor. The design tip speed ratio is usually defined between 7 and 9 for large wind turbines. As λ gets bigger, the blade will be more slender and flexible, which has the advantage of reducing the load. However, it may cause interference problems between the blade and the tower at extreme wind conditions and the blade rotation speed may need to be increased to achieve the targeted output power. On the contrary, a smaller value will contribute for the blade being thicker and cause greater axial thrust forces.

The rotor geometry is periodic so it is recomened to use cylindrical coordinates. To define the 3D blade, it is convenient to define an adimensional parameter for the direction normal to the flow direction, used to define how the airfoil will be altered in a certain position, and defined as

$$\mu = \frac{r}{R}\,, \tag{3.2}$$

where r is the distance do the blade axis of rotation and R is the radius of the turbine.

As the blade is finite, it was necessary to define a blade tip loss factor,

$$f = \frac{2}{\pi} cos^{-1}(e^{1/((N/2)(1-\mu)/\mu)\sqrt{1+(\lambda_{design}\mu^2)/(1-a)}})\,, \tag{3.3}$$

as it was first define by Prandlt and then modified by Glauert [17] in their aproach to the BEM theory. This definition depends on the flow induction factor a, which the exact value is given by

$$a = \frac{1}{3} + \frac{1}{3}f - \frac{1}{3}\sqrt{1 - f + f^2}\,, \tag{3.4}$$

making it necessary to calculate this values using a recursive method.

Wind turbine blades are designed with tapered and twisted shapes to obtain maximum power output with high efficiency. In [40] several discretizations for the tapper across the blade were proposed, having

all similar aerodynamic results, from which the best chosen to be applied in this work was

$$\frac{c}{R} = \frac{2\pi}{N} \frac{4\lambda^2 \mu^2 a'}{\sqrt{(1-a)^2 + (\lambda\mu(1+a'))^2}}\,, \tag{3.5}$$

where N is the number of blades in the HAWT and C_l is the lift coefficient of the airfoil for the angle of attack α defined. Which depends on another flow induction factor a'

$$a' = \frac{a(1-a/f)}{\lambda^2 \mu^2}\,, \tag{3.6}$$

which also takes in account the tip speed ratio and the position in the blade.

At the locations for which the chord was calculated, it was also calculated the blade twist ϕ , which is defined by

$$tan(\phi) = \frac{1-a}{\lambda\mu(1+a')}\,, \tag{3.7}$$

which is then applied by rotating the airfoil at that position by a pitch angle θ which also depends on the angle of attack on the blade as is defined by,

$$\theta = \alpha - \phi\,. \tag{3.8}$$

All these equations were programmed in a *python* [41] script that accepted regular airfoil files, being possible to define the airfoil in up to five radial sections of the blade, and where the positions that were not defined by the user would be a result of a interpolation between the coordinates of the two prescribed airfoils that were closer to that location. A sample python script is included in the Appendix A

To have a preliminary estimate of the blade, the same input values as in [40] were used, which are summarized in Tab. 3.1. This code will then define a list of coordinates that are representative of the blade, a list of 3D points that represent normal cuts on the blade external shell that are possible to interpolate in order to obtain accurate geometry of a generic HAWT blade, as shown on Fig.3.3.

Table 3.1: Basic design parameters of a HAWT blade.

Rated wind Speed	$12.1m/s$
Diameter	$94.8m$
Number of Blades	3
λ_{design}	7.5
Rotor Speed	$15.11rpm$
Angle of Attack	2

3.2 Mesh Generation

The definition of the mesh is of major importance. A good mesh, should have enough definition to be realistic, but not too much so that the the computational time requires is acceptable for a work of this nature. To have an acceptable mesh that produced good results in OpenFOAM, it was necessary to evaluate the capabilities and possibility of integration with the solver software of several mesh generators.

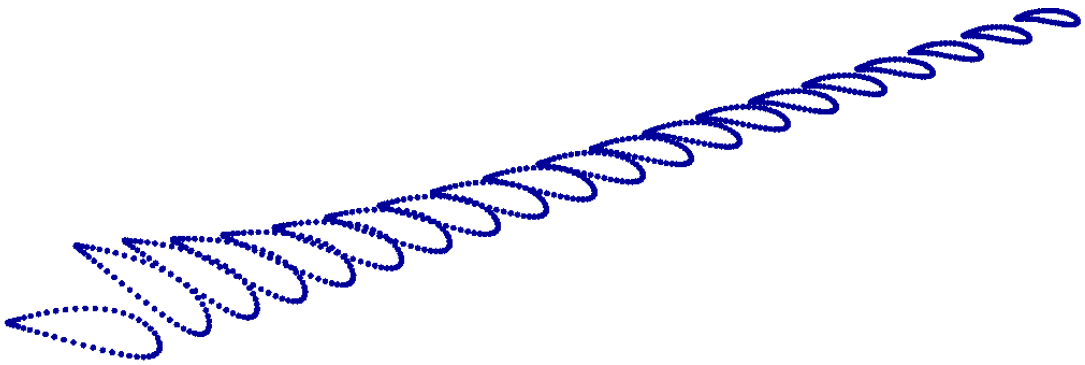


Figure 3.3: Point cloud of the blade surface generated by the *python* script.

Each of these mesh generators will be briefly described next.

3.2.1 *Gmsh*

Gmsh is a 3D finite element grid generator with a build-in CAD engine and post-processor [42]. It is a open source software, widely used in academic applications and, for this reason, it was the first option to use as a mesh generation tool for this work. It is built around four modules: geometry, mesh, solver and post-processing, and the specification of any input to these modules is done either interactively using the graphical user interface (GUI) or in batch mode with ASCII text files using *Gmsh*'s own scripting language, being in this case ideal for the automation algorithm that is intend to be used.

It was quite simple to generate basic meshes, as illustrated in Fig.3.4, with the appropriate parametrization using the scripting language and some file editing with python. However, the difficulty of controlling the element sizes as well as the difficulty of generating a structured mesh, which seems to be only possible through extrusion of 2D profiles, determined that this tool was not appropriate for this work.

3.2.2 *blockMesh*

This is the mesh generator that is included with recent versions of OpenFOAM. As other OpenFOAM tools, the user interface is done by a single text file, as there is not a GUI, called *blockMeshDict* where the grid is defined by the eight nodes of every block in the mesh. Then, it is possible to define different types of edge for the block, being feasible to define with precision every part of the geometry that is intended to be modeled. The meshes generated by this tool are always structured and have the added advantage of being exported directly in OpenFOAM format, not needing an additional step for conversion, which all the other generators need.

The OpenFOAM mesh format, which is called *polyMesh*, is based around faces and each face is assigned a ”owner” cell and ”neighbour” cell so that the connectivity across a given face can simply be described by the owner and neighbour cell labels. In the case of boundaries, the connected cell is the owner and the neighbour is assigned the label -1. The mesh is described in five text files, which describe points, faces, owner and neighbour cells and boundaries. Boundaries and blocks are defined as patches in OpenFOAM, which must be described when defining the mesh.

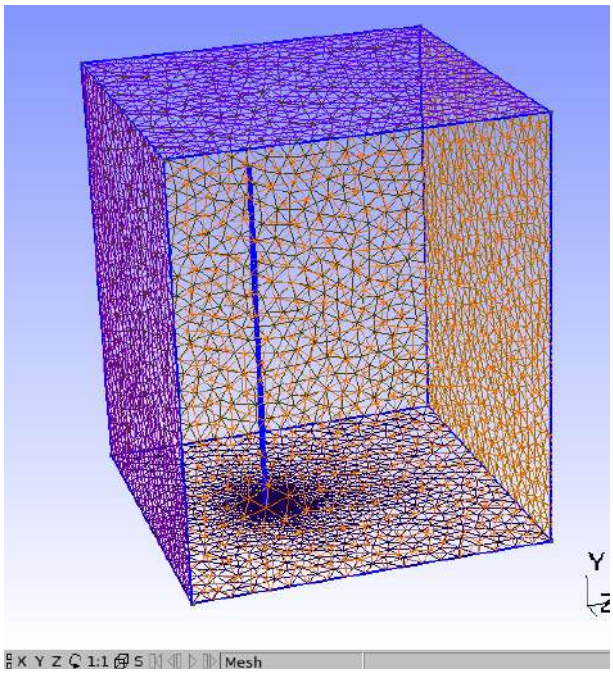


Figure 3.4: Basic mesh of rectangular wing using *Gmsh*.

This was the tool that would probably produce the better results. It was used to define simple 2D meshes that were used as test cases but, as OpenFOAM itself is not very user friendly. Defining the complete 3D mesh would take longer than desired for a six months assignment and so this solution was not explored any further.

The initial studies were performed using a 2D mesh of a beam, this mesh was generated using *blockMesh*, and consisted of a structured 3D mesh with only one element deep as seen in Fig.3.5. This mesh had a structural counterpart, which is presented in Fig.4.4(a), that was used to test the structural solver. Later on, both meshes were used in the tests and development of the fluid structure interaction solvers.

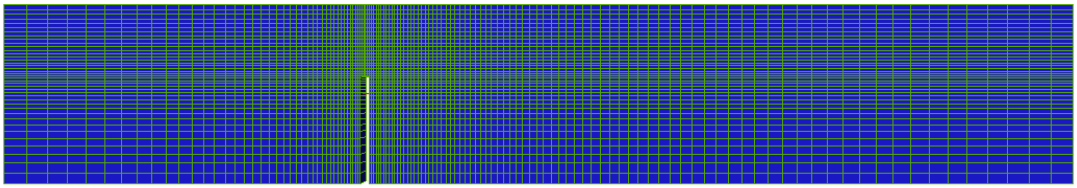


Figure 3.5: Simple mesh of beam in cross flow using *blockMesh*.

3.2.3 ANSYS ® ICEMCFD

As none of the open-source solutions seemed to give satisfactory results, it was decided to use ICEM-CFD 3.2.3, the mesh generator provided by ANSYS®. This software was chosen because it is user friendly, it has proved results in both academy and industrial research and has a broader user base than any of the other solutions. As the solutions previously discussed, ICEMCFD also has the possibility of

interact through scripting language, based on the Tcl/Tk codes, which can be written directly on a file or recorded with the use of the replay control window on the program GUI.

With this software, it is possible to define complex geometries quickly, by defining lines and surfaces, and using the multi-block strategy to obtain more control over the element sizing and the geometry adaptation. The approach chosen to define the mesh was the same as in the *Gmsh* case, it was created a small srcipt in *python* that writes the ICEMCFD appropriate commands into a text file and then this text file is run in ICEMCFD in order to export the mesh. All the parameters are defined in python file, sending different coordinates to the text file in order to modify the mesh automatically.

In ICEMCFD, mesh are defined by inputing points, which can be combined to generate different types of lines. This lines can then be joined together to produce surfaces and there are also some predefined surface shapes available. There are several ways to define the volumes, but the better one is to define a space close by surfaces and then define a material point inside.

To obtain controls over the mesh sizing, it is necessary to use blocking and define several block in order to have a reasonable approximation of the domain. There are then a lot of functions and different ways to achieve the mesh sizing and structure desired.

After selecting ICEMCFD as the mesh generator, it is important note that the geometry was defined directly in the mesh generator, and was not created in CAD, as this is better for automation as it reduces one step, also reducing the computational time. As already stated, the interface method for communicating with the software was chosen to be by scripting language, in order to simplify a future optimization process.

A python script was developed and functions were defined to automatically generate the different commands, such as straight lines, splines and surfaces. This script was different for the solid and fluid meshes and the commands generated were then saved to ICEMCFD replay files. These codes generate the geometries defined for the fluid and solid domains and define the blocking for each of the geometries. The mesh is then computed and exported in *Fluent* format, and is copied to the case folder and converted to the OpenFOAM format using *fluent3DMeshToFoam*, which is the default mesh converter from OpenFOAM to convert thee-dimensional *Fluent* meshes. All this process was programed to be automatic.

Using the capabilities of the software used and the code created, the point cloud created previously was uploaded, and using these points the blade starts to be formed by defining splines across the all points of the blade, producing the lines in green, visible in Fig. 5.14. The surfaces were programed to be created automatically from these lines, using as much automation as possible in order to be possible to change the geometry at any time without having to redefine all the surfaces.

A sample *python* script is included in appendix A.

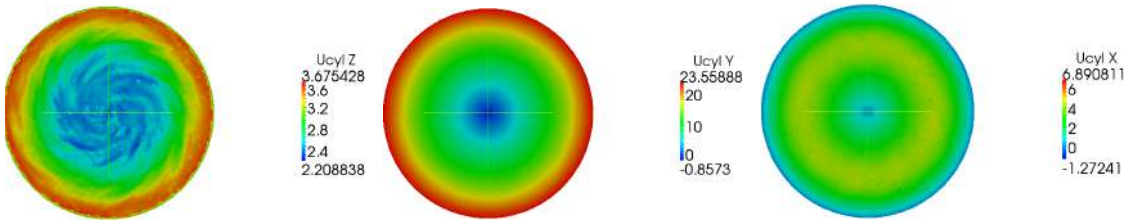
3.3 Post Processing

Most of the post-processing of the simulations is done using ParaView Viewer software [43], which is an open-source, multi-platform data analysis and visualization application. Using OpenFoam script

paraFoam, which creates a ParaView file with all the information from the case, users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or recurring to batch processing capabilities. This software was used for results presented as pressure and velocity distribution, vector fields, as well as to show the stream lines of the flow.

Although the software had the capabilities to do this, this software was only used in the interactive mode, where the calculation consume a lot of time and the possibilities are too limited, motivating the need to create separated small scripts to perform some of the post processing operations.

The first code created was to convert the Cartesian velocity field obtained from the simulation to a cylindrical referential. This code receives the coordinates of the mesh points from OpenFOAM and, according a predefined coordinate system, calculates the cylindrical coordinates in the mesh. After this, the code receives the velocity files, which can be for one specific time step or for all the time steps that were recorded during the simulation, and converts the velocity in radial, tangential and axial velocity, and writes this information into a text file called Ucyl, which can be read and incorporated in the ParaView model. This code was developed as a basic solver for OpenFOAM, which means that it uses standard tools form the main source code to import and export the data. It was tested in a simple tubular case, with an axial entry flow of $3m/s$ and a constant angular velocity of 1 rpm, and the results are presented in Fig. 3.6 and the code is presented in appendix B.



(a) Axial velocity (b) Tangential velocity (c) Radial velocity

Figure 3.6: Velocity fields as calculated by the code developed.

To evaluate the quality of the design, it was necessary to calculate the power that could be extracted from the blades. The method achieved consists of using an OpenFOAM library that calculates forces and moments on a determined surface for each time step, which can be viscous or pressure forces, saving the results to a data file, which means it would give all the information needed to calculate the mechanical power generated by the blades. The information obtained has then to beprocessed: the correct couples of forces (viscous and pressure based) were added the power obtained from a rotating body was calculated using moment generated by the flow on the blades (Torque - T),

$$P = T * \Omega \,, \tag{3.9}$$

where Ω is the rotation speed in rad/s. In addition, the drag/thrust forces were also calculated, which are important as the structural elements in the HAWT must be able to support it, and the power coefficient (C_P) which served as a comparison between the results obtained, and the maximum theoretical possible

and the values which are reachable with todays top commercial solutions. To present the results as clear as possible, it was created a small code which make the calculations and generate the plots of the parameters calculated throughout the simulations, this code is also available in appendix C.

Chapter 4

FSI Solver Definition

The main tool used in this project was OpenFOAM (Open Field Operation and Manipulation) [21], a C++ toolbox designed for the development of customized numerical solvers, including pre and post-processing utilities for the solution of continuum mechanics problems, namely computational fluid dynamics (CFD). In this specific work, only steady state and incompressible solvers will be used, but OpenFOAM capabilities are much bigger, being possible to find, between official releases and user contributions, almost every type of solver that is possible to find on similar commercial software. The code is released as free and open source software under the GNU General Public License, becoming a great tool for academic research.

For this work, it was chosen to use the version 1.6-extended of the software [44], this is the last purely development version and, even though most of its functions have been passed to more current versions, it is still the most commonly used by developers. In this version, it possible to use predefined solvers, fluid models and boundaries, as well as define new ones or improve the existing ones, making them suitable for the problem in hands, and add them to the source code or distribute them back to the community. OpenFOAM does not have a GUI, which means that the user will need to have at least some basic knowledge and experience of CFD to be able to define a case. Furthermore, it is also very difficult to interact with this software without having some knowledge of C++ and Linux shell environment, making the OpenFOAM learning curve severely longer than its commercial competitors.

In this software, a case is defined by three main folders that contain text files in which the entire case is described: the folder *constant*, where the mesh and material properties are described; *system*, where the solver, discretization schemes and the solution correctors are defined; and finally *0*, where the boundary and initial conditions are specified. After the simulation, the results are stored in folders named after the time step recorded.

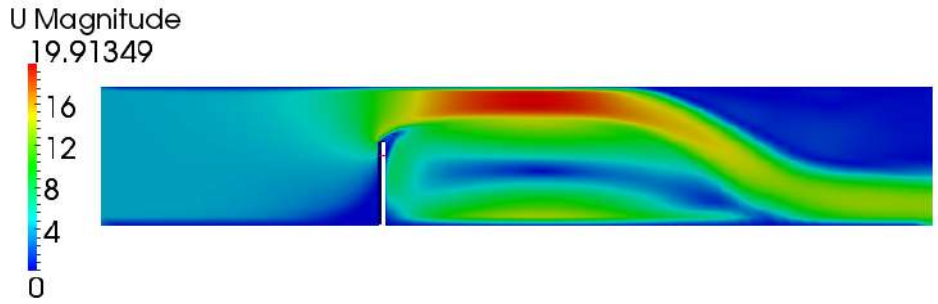
In Sec.4.1, a brief description of the relevant solvers available in the version 1.6-extended of OpenFOAM is given. These solvers were used as building blocks for the development of two specialized solvers for the aero-structural analysis of HAWT blades, described in Sec.4.2

4.1 Standard OpenFOAM Solvers

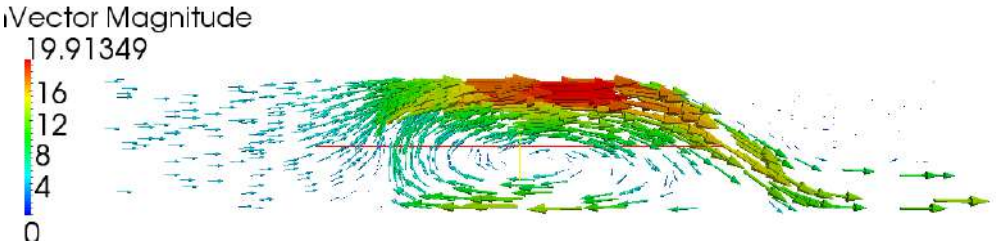
4.1.1 *icoFoam*

icoFoam solves the incompressible laminar Navier-Stokes equations using the PISO algorithm [45]. The code is inherently transient, requiring an initial condition, such as zero velocity, and boundary conditions. The *icoFoam* code can take mesh non-orthogonality into account with successive non-orthogonality iterations. As the flow is always laminar, it is only necessary to define the boundary and initial pressure and velocities.

Given that the solver is inherently transient and does not have any dissipation due to turbulence, the solution oscillates during a long time. Although the viscosity, geometry and boundary conditions remain the same, the fluid reaches faster velocities. This happens because flow is laminar, which means that there will not be kinetic energy dissipated by turbulence, making the velocity larger. The solution presents a recirculation bubble which is something to expect considering that the transition is very brusque. In this case, the bubble is encapsulated by the flow.



(a) Magnitude



(b) Vector Field

Figure 4.1: Velocity around a 2D beam using *icoFoam*.

4.1.2 *simpleFoam*

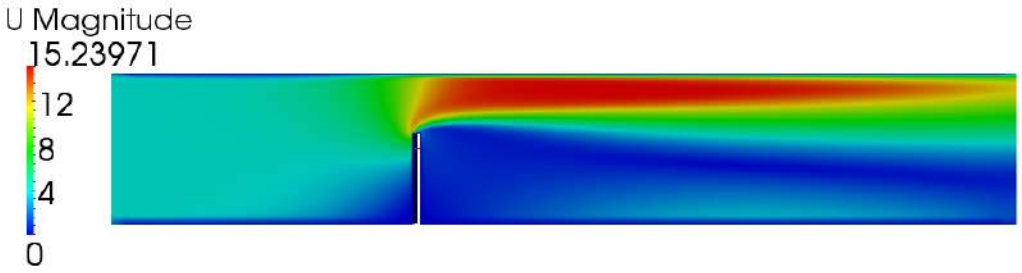
simpleFoam is a steady-state solver for incompressible and turbulent flow. As most subsonic CFD solvers, *simpleFoam* is based on the Navier-Stokes equations, which for a single-phase flow with a constant density and viscosity can be described by

$$\nabla \cdot \left(\rho \vec{U}\right) = 0 \tag{4.1}$$

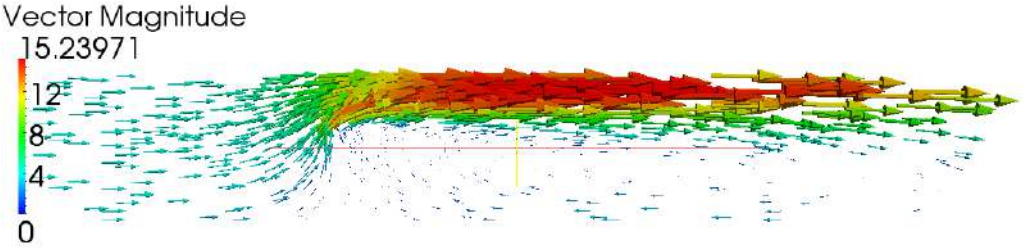
$$\frac{\partial U}{\partial t} + \nabla \cdot (\vec{v}\vec{v}) - \nabla \cdot (\nu \nabla \vec{v}) = -\nabla p \tag{4.2}$$

The solution of this system of equations is not straightforward because an explicit equation for the pressure is not available. One of the most common approaches is to derive an equation for the pressure by taking the divergence of the momentum equation and by substituting it in the continuity equation. The equations are then solved using the SIMPLE algorithm [46].

Using the mesh presented in Sec.3.2.2, a series of tests using different OpenFOAM solvers were performed. In a first step, the flow consists on an uniform velocity inlet at $4m/s$, giving a Reynolds number of 24000, wall boundaries on the top and bottom, a wall on the beam portion and fixed constant total pressure at the outlet. The turbulence model and boundary parameters were also defined, and for this case it was used the $k - \epsilon$ model as is the most commonly used.



(a) Magnitude



(b) Vector Field

Figure 4.2: Velocity around a 2D beam using *simpleFoam*.

The results obtained in Fig.4.2 show that this solver is very stable, as it achieved convergence quickly, and the results obtained seem physically correct. The flow accelerates near the beam, and after the beam there is a recirculation bubble that extends trough the rest of the domain, which is something to expect considering that the transition is very brusque and the Reynolds number is high.

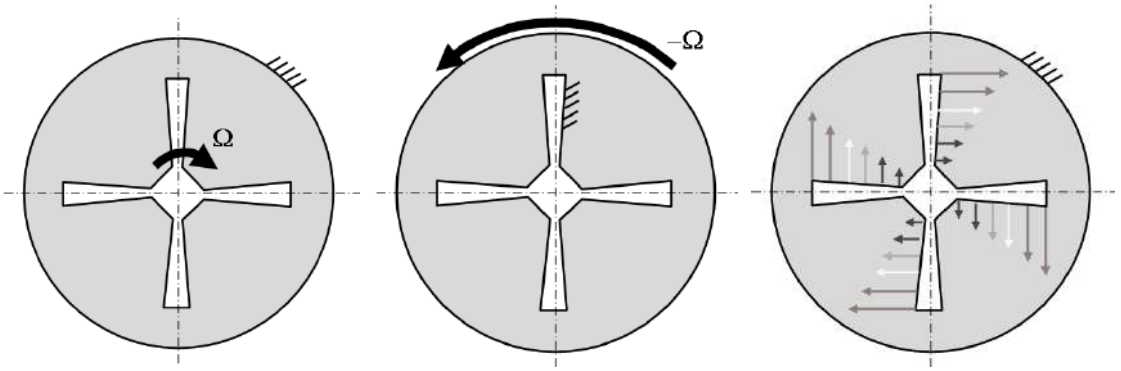
4.1.3 *simpleSRFFoam*

After trying to obtain results by applying a series of rotating velocity boundary conditions, using *simpleFoam*, this revealed to be impossible. In order to obtain better results for the HAWT blades simulations, and considering the limitations in the computing power available, it was necessary to find a solution that brought better results and was computationally cheap.

The solution to this problem was obtained by using a single rotating frame (SRF) solver, since a rotating frame of reference is often used to model flows in rotating machines. In this solver the flow is

unsteady in an inertial frame because the blades or rotors sweep the domain periodically as shown in Fig. 4.3(a). However, it is possible to perform the calculations in a domain that moves with the rotating coordinate which is fixed on the rotating part, and for this the angular velocity must be set as symmetric to the real velocity (Fig.4.3(b)). The flow is steady relative to the rotating frame, which reduces the computational cost needed for a more accurate analysis. When transposing the results for the inertial referential, it appears as the blades are moving(Fig.4.3(c)).

This approach is appropriated when the flow at the boundary between the rotating parts and the stationary parts is weakly affected by the interaction. It provides a reasonable time-averaged simulation result for many applications. Rotating frames do not physically rotate anything and therefore do not show transient effects due to the real motion and the Coriolis and centrifugal forces are treated as body forces. Any problems where transient effects due to rotor-stator interaction are small are candidates to use the rotating frame of reference approach. Since this was the case in this work, this steady-state solution was considered appropriate.



(a) Inertial referential (b) Rotating referential (c) Equivalent motion of the blades

Figure 4.3: Single rotating frame of reference [47].

4.1.4 *stressedFoam*

stressedFoam is a transient or steady-state segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses. It solves for the displacement vector field $\vec{u}$, also generating the stress tensor field σ, using a simple linear elasticity structural analysis code. The differential form of the force balance for the solid body element is then given by

$$\frac{\partial^2 \vec{u}}{\partial t^2} + \nabla \cdot \sigma = \rho \vec{f}, \tag{4.3}$$

where ρ is the body density and $\vec{f}$ is the body force.

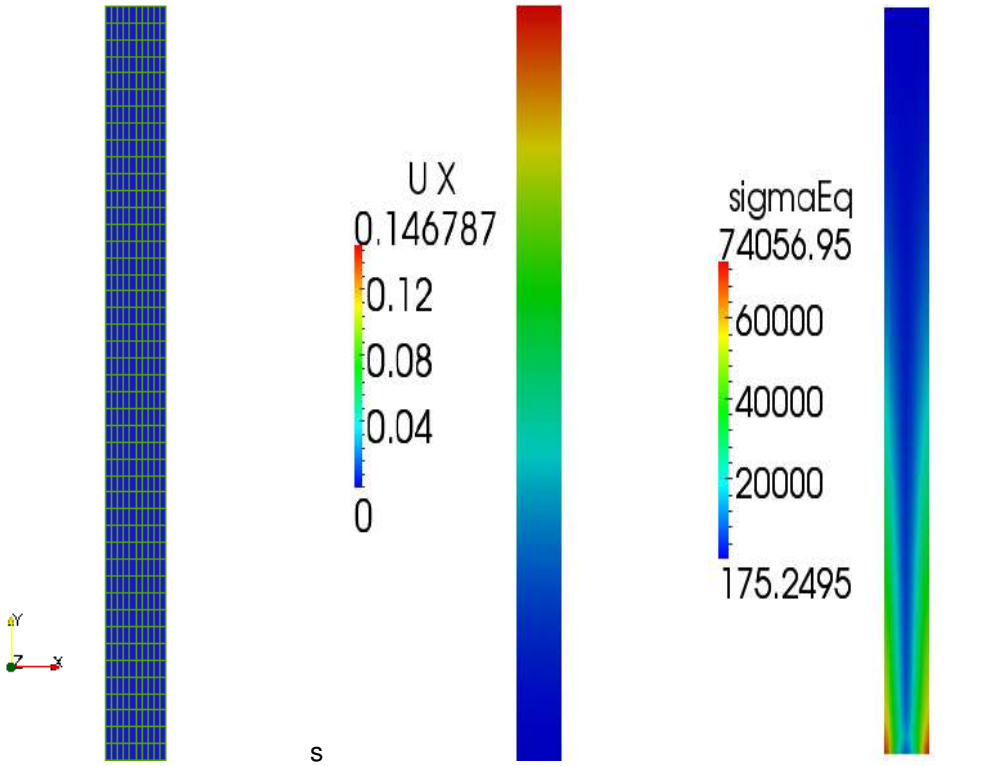
Considering the Hooke's law and the common definition of strain, Eq.(4.3) becomes

$$\frac{\partial^2 \vec{u}}{\partial t^2} + \nabla \cdot [\mu \nabla \vec{u} + \mu (\nabla \vec{u})^T + \lambda I tr(\vec{u})] = \rho \vec{f}, \tag{4.4}$$

where $\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}$ and I is the unit tensor [48]. After discretization, Eq.(4.4) is solved for each

element of the structural mesh.

To evaluate the capabilities of the structural solver, a rectangular beam with Young Modulus of $2MPa$, density of $1000Kg/m^3$ and ν of 0.3 was used. This beam was loaded with a constant pressure of $100Pa$ in the X direction and clamped on the bottom part, meaning that the beam is subjected to a constant bending force. Fig. 4.4(a) shows the mesh used in this case and figures 4.4(b) and 4.4(c) show the displacement in the x direction and equivalent stress, respectively. This displacements correspond to the real deformation, which happens linearly along the x direction. For Von Misses equivalent stress, the results are also as expected, as the tension applied is constant around the 2D beam, the biggest stress concentrations have occurred near the clamping, while on the free top, the stress is in the same line with the quantities applied.



(a) Mesh (b) Displacement in x direction (c) Equivalent Stress

Figure 4.4: Structural simulation on a 2D beam using *stressedFoam*.

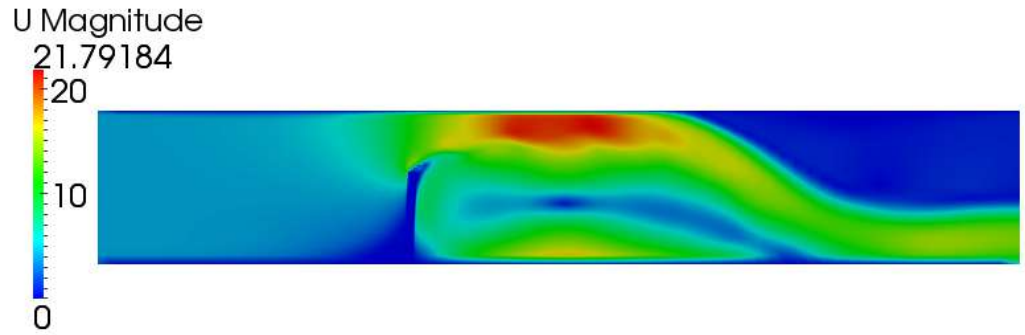
4.1.5 *icoFsiFoam*

icoFsiFoam is a fluid-structure interaction (FSI) solver for weakly coupled systems with small structural deformations and laminar fluid flow. The structural part is based on *stressedFoam*, thus limited to linear stress strain relationships and small deformations. The fluid part of the solver is based on the *icoFoam* and is limited to Newtonian fluids in incompressible, laminar flow. *icoFsiFoam* does not use the PISO algorithm used in *icoFoam*, but a SIMPLE-based algorithm with corrector loops specified by PISO parameters [49].

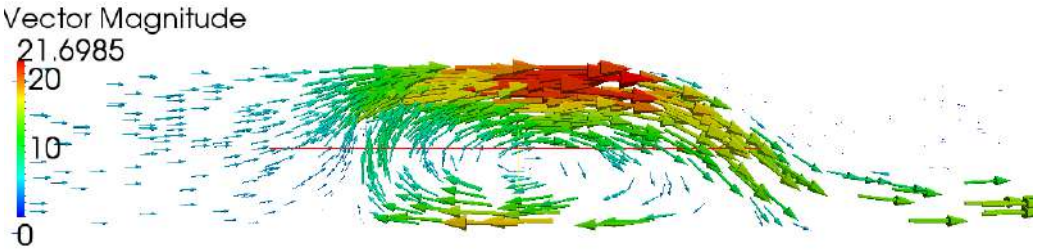
This solver consists, in fact, in having multiple solvers side-by-side, which means that each solver

uses its own mesh, with access to its fields, material properties, solver controls, etc. The fluid-Structure coupling is achieved through boundary condition updates. In this case, this coupling is achieved by the following loop: the pressure data from fluid is moved to structure; the structural equations are solved; the displacement of the solid mesh is then transfered to fluid mesh; the fluid flow is solved with mesh motion and the loop restarts.

To test this solver, the two mesh that were analyzed earlier were used, being the conditions the same as in the fluid cases presented, with the difference that in this case the walls corresponding to the beam were moving walls, instead of being fixed. The *icoFsiFoam* results illustrated in Fig.4.5 show that the fluid has a behavior similar to the case where *icoFoam* was used, but the differences to the turbulent case were even more evident. In this case, the velocity reaches even higher values as the beam bends, the disturbance to the flow is smaller, with the immediate visible result being an increase in velocity.



(a) Magnitude



(b) Vector Field

Figure 4.5: Velocity arround a 2D beam using *icoFsiFoam*.

4.2 User-Defined Solvers

4.2.1 *simpleFsiFoam*

Due to the major limitations of *icoFsiFoam*, as the impossibility to define a turbulent flow, it was necessary to develop a solver that could model the flow around a turbine blade more accurately. This was considered important due to the fact that the flow over wind turbine blade is always turbulent. Is was decided to develop *simpleFsiFoam*, a FSI solver based on *simpleFoam*. This solver takes advantage of the simplicity of *simpleFoam*, as well as the extra functionalities that it has over *icoFoam*.

As simplicity and flexibility were important, the fluid-structure coupling was achieved in the same way that is done in *icoFsiFoam*, which means that there is still two different meshes, one for the fluid and the other for the structural solver. The meshes do not need to have the same amount of elements in the correspondent boundaries, which is ideal for the coupled optimization process, as small variations can be accomplished from both meshes, without having problems with the coupled properties.

As described earlier, the partitioned approach is being implemented for this work, employing a staggered solution procedure wherein each domain. This approach was also used because of the compatibility with black-box solvers, which means that even though this solver has been developed to use *stressedFoam* and *simpleFoam*, any of the solvers may be replaced by a similar one, without much change in the code, simply replacing the part were the calculations are made.

As showed in Fig. 4.6, each iteration begins with setting the pressure around the solid mesh, using the last iteration fluid information. The solid part is then solved and the displacement is applied, deforming both of the meshes, and finally the fluid is solved. After each iteration, the solver evaluates if it is time to save the solution, as it is possible to define in *controlDict* the writing interval, and increments time, until the end time is reached.

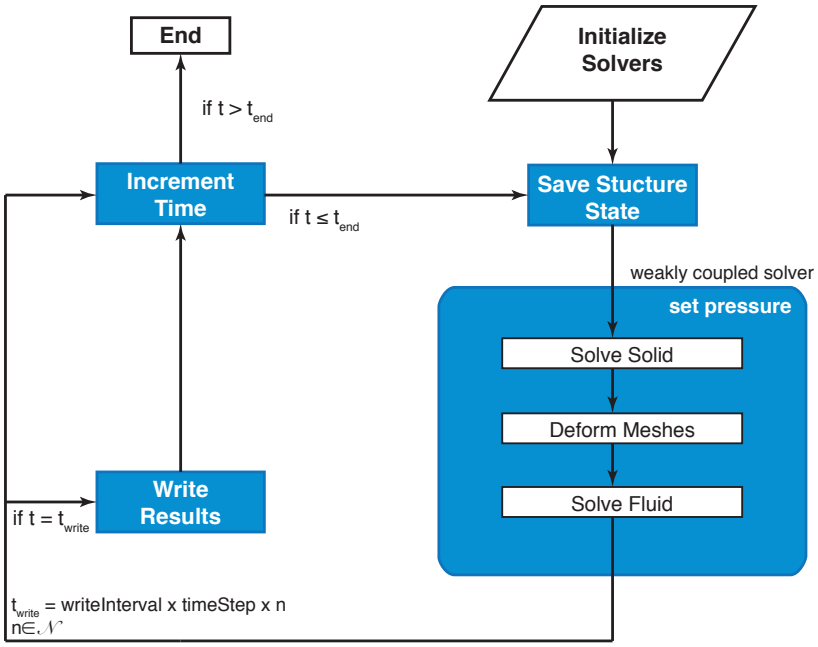


Figure 4.6: Fluxogram for the FSI solver.

To define a case, it is necessary to define the two cases in the same folder and define symbolic links between them, as is shown in Fig.4.7. These soft links linking together the settings for the fluid phase with the solid phase are necessary in order for the solver to get input to the solution of both the fluid and structural equation systems. The fluid-structure interface is defined after both the fluid and solid meshes have been created.

The basic layout is the same as in any OpenFOAM case, but much more information is required. In the *0* folder for the fluid is defined the initial and boundary conditions, using the *U* file to define velocity associated to each boundary; the *p* file to impose the pressure settings; and *nuT*, *nuTilda*, *k*, *omega* and

epsilon are used to define the turbulence properties depending on the turbulence model to use. The file *motionU* contains the conditions for the deforming mesh motion, which is a requirement for the FSI analysis. Using *movingWallVelocity* as the boundary condition in *U* at the interface, the motion of the mesh at the interface is implemented as a Dirichlet boundary condition for the fluid velocity. For the solid mesh, in *0* the only property that has to be defined is the initial displacement which is set on the *u* file.

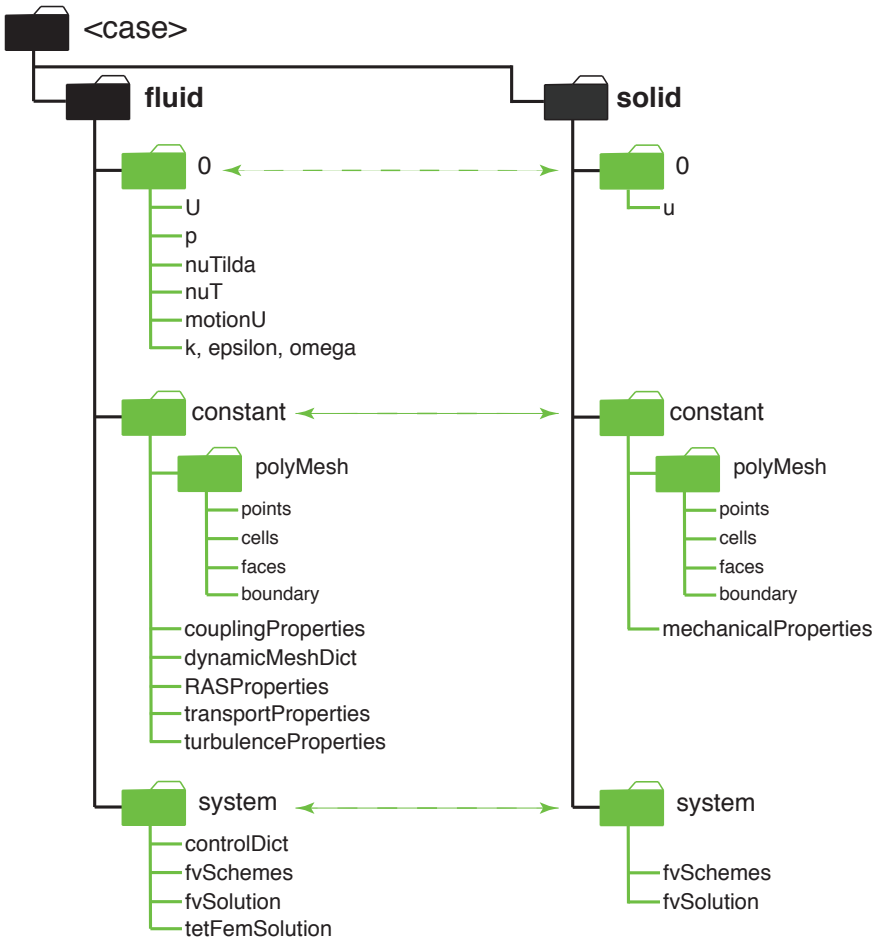


Figure 4.7: Schematic of the definition of a case for *simpleFsiFoam*.

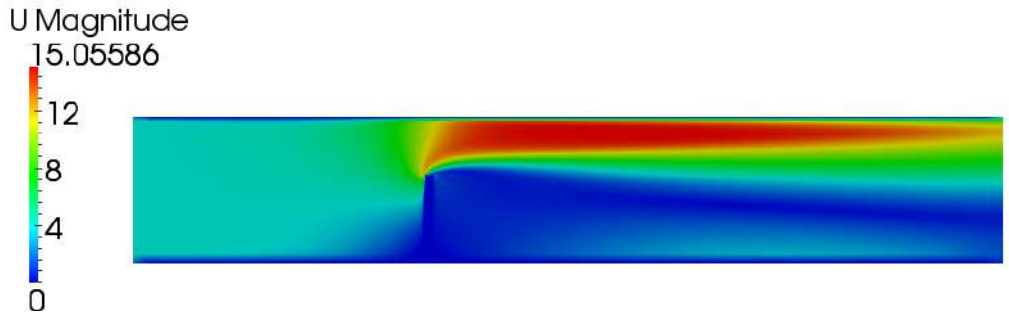
In the the *constant* directories most of the information contained is about the meshes and the material properties, as explained previously, the *polyMesh* folders contain all the necessary information about the meshes, as for the fluids transport properties, such as density and viscosity, are defined in the file named *transportProperties*, while the solids material properties like density, Young modulus, possion coefficient etc., are defined in *mechanicalProperties*. As the names denote, the turbulence properties and model are defined using the *RASProperties* and *turbulenceProperties* files. In the *constant* of the fluid part are also defined most of the settings governing the fluid-structure interaction are defined: in *couplingProperties*, are defined the coupled boundaries and the moving mesh patch; the settings for the solution of this patch motion are defined in *dynamicMeshDict*.

The *system* directories are where the solution procedure is setup. Settings such as end time, time step and time interval for output to be written are all defined in *controlDict*, also in this file is defined if

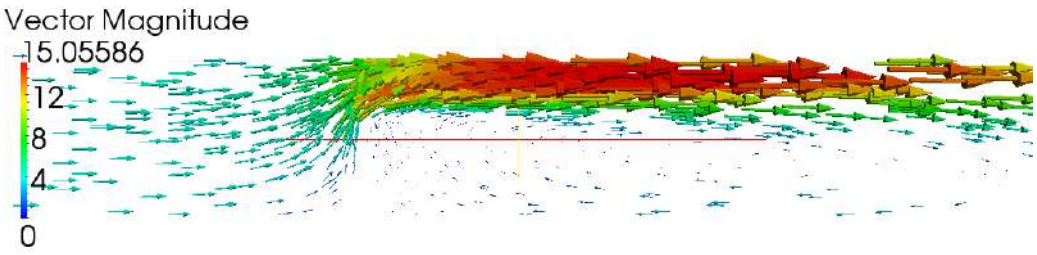
some none default library is going to be used, as is the case with the library that calculates the forces or the special libraries for turbo-machinery applications. The files *fvSchemes* contain information about which differencing schemes to use and the files *fvSolution* the settings for convergence tolerances.

In the development versions, OpenFOAM has a vertex-based, unstructured mesh motion solver, which is used in this case. This solver executes Laplace face decomposition and performs very well for many problems and it is easy to implement because only motion on the tracked interface is required, but it does have limited functionality. In particular, in its current form it does not function properly for parallel solutions and it does not support periodic (cyclic) boundaries. This is a major limitation to the use of OpenFOAM as a fluid-structure interaction tool for turbo-machines, as this machines usually have periodic geometries, which means that a big part of the computing power is wasted because of the impossibility to use periodic boundaries. Furthermore, in the special case of the horizontal axis wind turbine where 3D effects are very important, the analysis would really benefit from a finner mesh that could only be feasible with the use of parallelization.

The same case that was tested with *icoFsiFoam* was tested with the new solver. The results were similar but, as in the case where it was used *simpleFoam*, the decrease in velocity due to turbulence is evident.



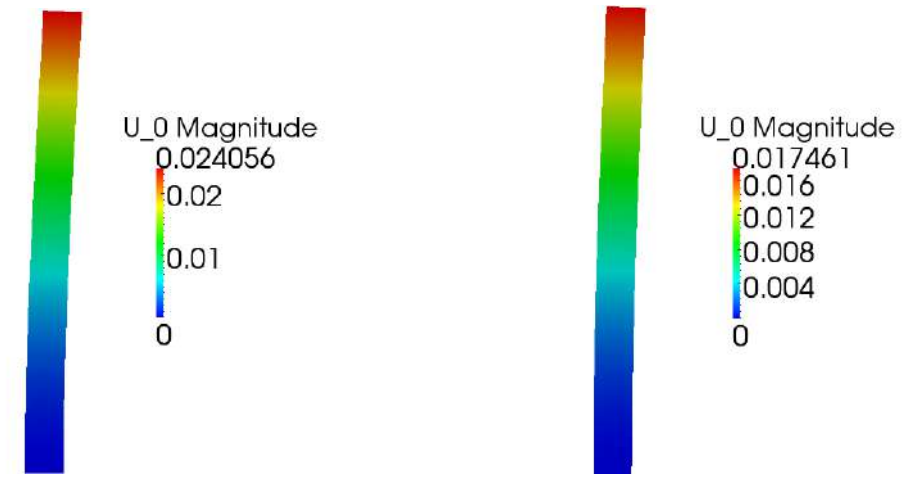
(a) Magnitude



(b) Vector Field

Figure 4.8: Velocity arround a 2D beam using *simpleFsiFoam*.

Comparing the results of the deformation of the beam between the two fluid-structure interaction solvers in Fig. 4.9, it is possible to observe that the laminar solver produces a larger deformation of the beam, as the velocity is bigger, so is the pressure differential between the two sides of the blade.



(a) *icoFsiFoam* (b) *simpleFsiFoam*

Figure 4.9: Displacement caused by the flow around a 2D beam.

4.2.2 *fsisimpleSRFFoam*

Using the same strategy as it was explained in Sec.4.2.1, it was developed another FSI solver for OpenFOAM, now using as base the Single Rotating Frame Solver(*simpleSRFFoam*). This was the solver used in the final simulations, which produced relatively good results. In the definition of the case, the only differences to the *simpleFsiFoam* case is that the rotational speed must be defined, in the *constant* folder, in a file called *SRFProperties*, and the boundary conditions for the velocity are now defined in the *Urel* file, where the velocity in the boundaries may be defined as relative or absolute. As stated before, all the calculations are made in the rotating referential and, only at the end, properties are transposed to the inertial referential. In this the absolute velocity is called *Uabs*.

4.3 Boundary Conditions

This section explores the properties of boundaries and their importance when working with OpenFoam. The subject of boundaries is directly related because their role in modeling is not simply that of a geometric entity but an integral part of the solution and numerics through boundary conditions or inter-boundary connections.

For the purpose of applying boundary conditions, the boundary has to be separated into a set of patches. One patch may include one or more enclosed areas of the boundary surface which do not necessarily need to be physically connected. There are three type of boundaries: base, primitive and derived.

The base type of patch described purely in terms of geometry or a data ”communication link”, this type of patches includes boundaries such as *wall*, which automatically defines fixed velocity although it may not be zero; *cyclic* and *wedge* which are never easy to define, as they have a lot of limitations, but work as periodic boundaries; *symmetry* that turns the boundaries into symmetry planes; and *empty* that reduces on dimension to the case, if the mesh is defined in the appropriate way and the parallel

areas in this patch have only one cell between them. This is the only way to define a non-3D case, as by default, OpenFOAM can only solve 3D meshes. There is one last type of base boundary which is called *patch*, and as is suggested by the name, this type does not define anything so the characteristics of the boundary must be defined for each property.

By defining special characteristics to a field variable, like fixed values, fixed gradient or mixed, the boundary becomes a primitive type. This is the way to define simple inlet and outlet conditions.

The derived type is an even more complex patch condition, derived from the primitive type, assigned to a field variable on the patch. This is the case of some of the boundary conditions that were studied in order to be able to simulate the flow around the wind turbine blades. These boundaries were studied with the objective of defining the rotating conditions that could simulate the case required.

All of these boundary conditions are add-on libraries in OpenFOAM 1.6-extended, which means that they are user defined and peer reviewed but not a part of the main release, being necessary to load the libraries through the *controlDict* file. With *profile1DfixedValue*, it is possible to define any profile desired to the inlet velocity, depending on the radius. This solution was considered interesting to define the inlet velocity of the domain. Unfortunately, the velocity field did not seem to converge using this method, and this was not a very practical approach for automation as if it was necessary to define a new spread sheet file with the information required, every time the velocity profile needed to be modified.

Another approach that gave better results was using the *cylindricalInletVelocity* boundary. The velocity is here defined with recurse to the definition of a rotation axis, axial velocity and angular velocity, the code then calculates the resultant velocity across all the surfaces that belong to the patch. The main problem with this boundary condition was that the it did not propagate to the interior domain as expected, only being true very close to the surface where the boundary was imposed.

The other derived boundary condition used was a requirement of the single reference frame solvers, and it is defined with the command *SRFvelocity*. This boundary works as the usual fixed value, but it is possible to define if the condition imposed is the relative or absolute velocity. If the velocity defined is relative, the rotation speed will be added to calculate the absolute velocity in this boundary.

4.4 Turbulence Models

OpenFOAM is distributed with a large library of Reynolds-Averaged Simulation (RAS) turbulence models. For incompressible solvers, there are available algebric models, such as Spalart-Allmaras one equation mixing-length model; as well as the usual two equation models: $k-\epsilon$ and its variations: Realizable, Re-Normalisation Group (RNG), Non-linear Shih, Lien cubic, Lien-Leschziner, Lam-Bremhorst and Launder-Sharma; $k-\omega$ and $k-\omega$-SST; laminar which is a dummy turbulence model for laminar flow and a few others.

Furthermore, considering the open architecture of OpenFOAM, it is also allowed developers to add turbulence models that can be selected by the user.

The most commonly used turbulence models is the $k-\epsilon$ and Spalart-Allmaras, have their theoretic formulation is presented next.

4.4.1 $k-\epsilon$ Model

The $k-\epsilon$ model for turbulence is based on kinetic energy, using two equations [50],

$$k_t = \alpha(\frac{k^2}{\epsilon}k_x)_x - \epsilon \tag{4.5}$$

and

$$\epsilon_t = \beta(\frac{k^2}{\epsilon}\epsilon_x)_x - \gamma\frac{\epsilon^2}{k} \tag{4.6}$$

where $k(x,t)$ is the turbulent kinetic energy, $\epsilon(x,t)$ is the rate of dissipation of the turbulent energy and α, β and γ are positive constants.

4.4.2 Spalart-Allmaras Model

This one-equation model is given by [51]

$$\frac{\partial\tilde{\nu}}{\partial t} + u_j\frac{\partial\tilde{\nu}}{\partial x_j} = c_{b1}(1-f_{t2})\tilde{S}\tilde{\nu} - \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((\nu+\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_j}\right) + c_{b2}\frac{\partial\tilde{\nu}}{\partial x_i}\frac{\partial\tilde{\nu}}{\partial x_i}\right] \tag{4.7}$$

and the turbulent eddy viscosity is computed from:

$$\mu_t = \rho\tilde{\nu}f_{v1} \tag{4.8}$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \tag{4.9}$$

with

$$\chi = \frac{\tilde{\nu}}{\nu} \tag{4.10}$$

where ρ is the density, $\nu = \mu/\rho$ is the kinematic viscosity, μ is the dynamic viscosity and c_x are the model constants.

Chapter 5

Wind Turbine Definition and Simulations

This chapter presents all the steps taken towards getting to the current stage of the design. Defining a complex case in OpenFOAM can be very tricky, due to the lack of a real support structure, as there is not an official forum or even an help file. Most unexperienced users usually choose to define really basic mesh, like the one for which the results were presented in the last chapter, or define very simple cases, usually using OpenFOAMs test cases. The approach in this case was not that, as the objective was clear, to simulate the fluid-structure coupling on the blades of a HAWT. The way to get there was not as trivial and every single option available had to be evaluated, considering time and computational limitations. To do this it was necessary to go through an iterative process between the mesh and the simulation definitions as it was not possible to predict if a mesh would work or not.

The simulation results that were considered important to reach to the final state of the mesh and the simulation will be presented in this chapter. While developing the simulation qualitative analysis were mainly made and only in the final solutions the focus was directed to quantitative analyze.

5.1 Fluid Domain Definition

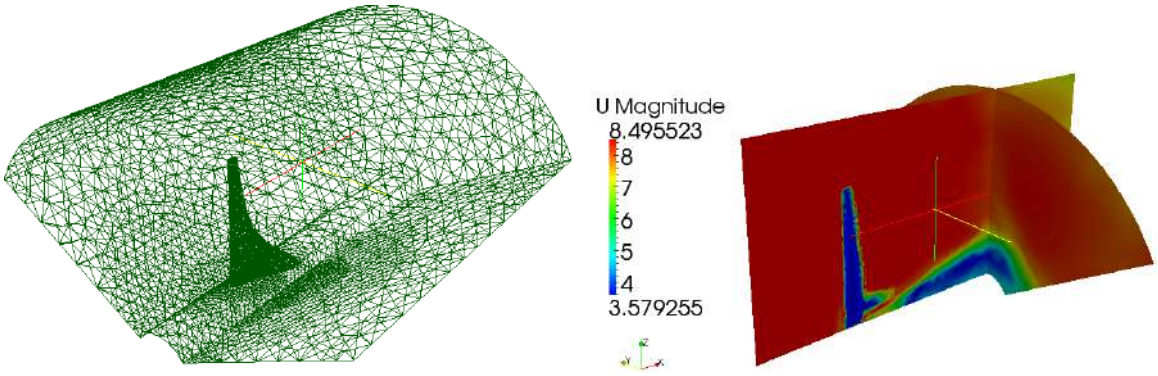
After defining the parameterization of the blade, the first challenge was how to define the fluid domain. Considering a three blade rotor, the first obvious choice was to define a 120° section of a cylinder, giving enough space to blade so that the boundaries and the wake do not affect the flow around the blade. At this point, it was considered that the central part should also be modeled as a cylindrical shaft to which the blade was connected.

The first mesh obtained is shown in Fig.5.1(a). Although the blade is defined in the same way as presented in Chapter 3, the distribution of the chord on the blade is not the same. This distribution, although it is considered the best in [4], gave a sail like shape to the blade, but this was considered acceptable as an initial case.

In this initial simulation, the velocity inlet had a constant value of $10m/s$ and all the other external

boundaries were modeled as pressure outlets. The turbulence was set to zero, and the solver used was *simpleFoam*. The velocity field obtained is shown in Fig.5.1(b). While these results were not particularly good, they were a starting point to analyze what was important to do next.

First of all, the results for velocity clearly had errors because of the size of the mesh, it was necessary to have a better definition to capture the boundary layer of the blade in more detail. The geometry of the blade had to be altered as this arrangement produced a recirculation bubble that is trapped near the blade, affecting the flow around the blade. It also showed the need to work on the boundary conditions, as even though the solution seemed steady (Fig.5.2), the results did not seem correct.



(a) Mesh (b) Velocity magnitude

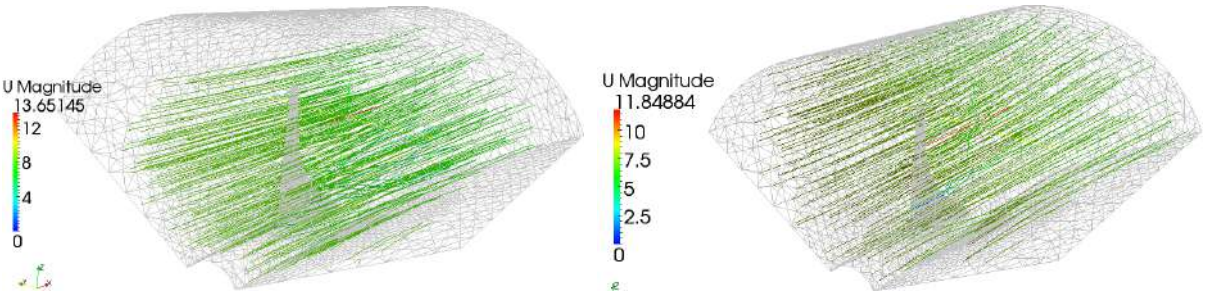
Figure 5.1: Initial HAWT blade simulation: mesh and velocity field.

5.1.1 Turbulent Inlet Conditions

Prescribing realistic inlet conditions is as important as it is difficult. Selecting a suitable turbulence model for turbomachinery simulations can be a challenging task. For a design iteration type of simulation a common choice is the one equation algebric model by Spalart-Allmaras [51]. This model has become popular due to the many inherent problems in more refined two-equation models, as $k-\epsilon$ and $k-\omega$. The biggest advantages of this model are its robustness and the fact of rarely producing of completely unphysical results. However, in more difficult cases, like separating or rotating flows, or flows strongly affected by secondary flows, the two equation models are better.

As the simulation did not account for the effect of turbulence in this flow, the first analysis to be made is to evaluate this effect. In this simple simulation, where the conditions were the same as in the case previously described with the alteration that the inlet was now turbulent using the $k-\epsilon$ model. As it is possible to see in Fig. 5.2, the maximum velocity decreases in the case with the turbulent inlet, stabilizing the flow and dissipating the recirculation bubble.

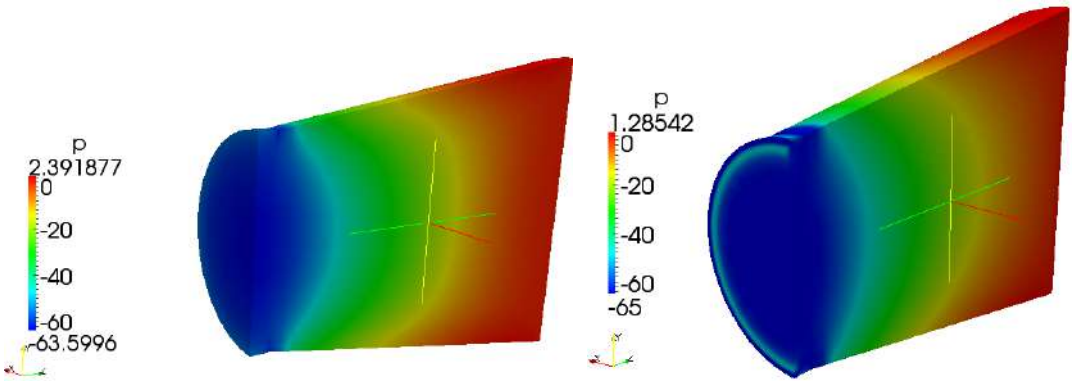
The simulations presented in this section had three main objectives: to evaluate if the Spalart-Allmaras turbulence model was reliable in OpenFOAM for turbulent inlet conditions; to evaluate the rotation inlet conditions presented in Sec.4.3; and to evaluate how did this boundary conditions worked in meshes generated in ICEMCFD. For this, it was used a basic diffuser mesh generated in blockMesh and a equivalent unstructured model generated in ICEMCFD.



(a) without turbulence (b) with turbulence

Figure 5.2: Initial HAWT blade simulations: comparison of turbulence conditions.

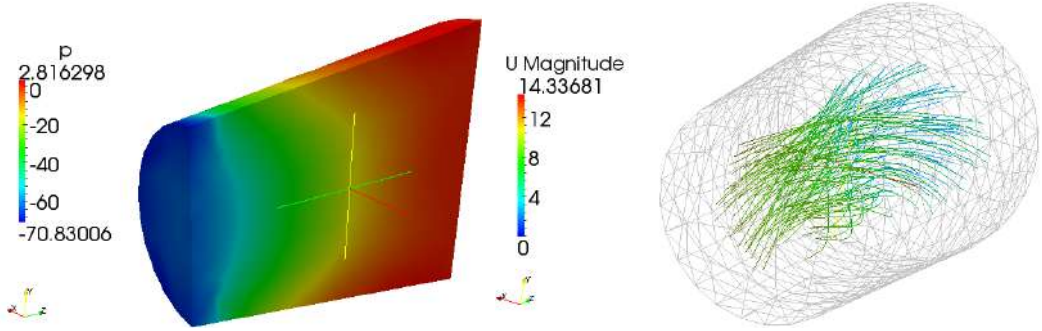
From the results in Fig.5.3, it is possible to understand that, in terms of pressure, there is not a clear difference between the two turbulence models, but when using mesh generated in ICEMCFD, with the rotating inlet conditions, as in Fig. 5.4 and 5.5, only the cases where the Spalart-Allmaras model converged.



(a) k-ϵ (b) Spalart-Allmaras

Figure 5.3: Pressure distribution in a diffuser using different turbulence models.

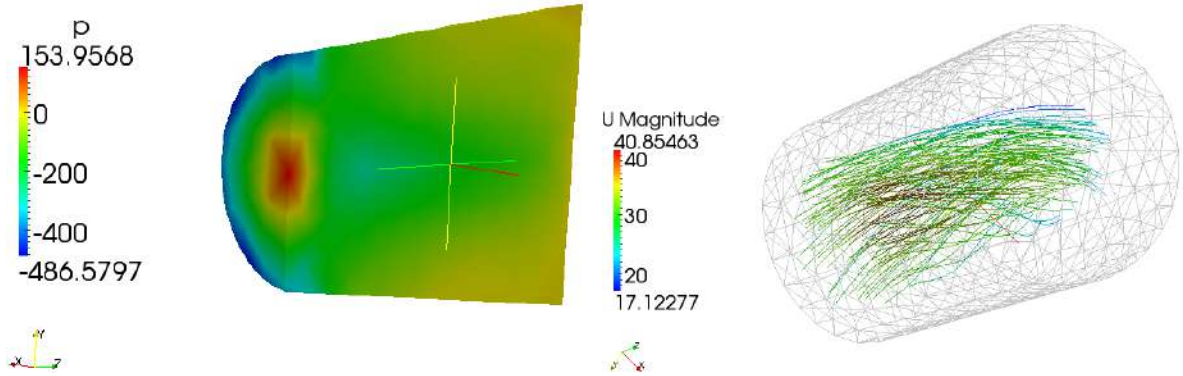
In the case presented in Fig. 5.4 it was used the *profile1DfixedValue* inlet condition. This condition presented good results, for the stream lines and pressure distribution. It is possible to understand that the flow is indeed rotating, and the results seem physically consistent with the flow on a diffuser.



(a) Pressure with *profile1DfixedValue* (b) Velocity with *profile1DfixedValue*

Figure 5.4: Simulation in a diffuser using *profile1DfixedValue* inlet conditions.

To test how would *cylindricalInletVelocity* perform in a similar case, it was used the same mesh and the axial and tangential components of the velocity were set. This also produced good results, as shown in Fig.5.5. In this case the results are not comparable with the previous as the inlet condition are totally different, but it serves as confirmation that this boundary condition would work on an unstructured mesh exported from ICEMCFD.



(a) Pressure with *cylindricalInletVelocity* (b) Velocity with *cylindricalInletVelocity*

Figure 5.5: Simulation in a diffuser using *cylindricalInletVelocity* inlet conditions.

The *cylindricalInletVelocity* boundary was chosen to continue because it is much more efficient to use then the other cylindrical inlet condition. In this case it, was only needed to define axial and angular velocity or, in alternative, the three cylindrical components of the velocity, while for *profile1DfixedValue* it was needed to define a complete file with the velocity profile in several points of the radius. This could be useful for other applications as it would be possible to define more sophisticated profiles but, in this case, it would only make the automation unnecessarily more complex.

5.1.2 Periodic Boundaries

As stated before, defining cyclic boundaries in OpenFOAM is not easy, but trying to avoid using the complete domain was a priority so it was decided to try using *symmetryPlane* boundaries and parameterizing the blade as a two bladed rotor. The mesh obtained, presented in Fig. 5.6, also took in account the better definition on the mesh that was considered necessary from the simulations on the initial mesh.

The simulations with this new mesh also served to test influence of the time step used in the simulations. In this matter, as it is explicit in Fig.5.7, using a time step of a thousand times smaller, the result is the same, as this a steady-state solver, as long as the Courant number remains small, near one, the time step counts only as an iteration counter.

In these simulations, it was also possible to understand that even though the simulations worked, as the stream lines presented in Fig.5.7 are showing the flow movement required, they were not very stable and usually the simulations presented errors after some time.

So, as this also did not seem a good option, it was decided to explore further the possibility of using a periodic mesh. To define a periodic mesh in OpenFOAM it is necessary to have the elements that are connected numbered in a specific way. If the number of elements that is in one of the periodic sides is n

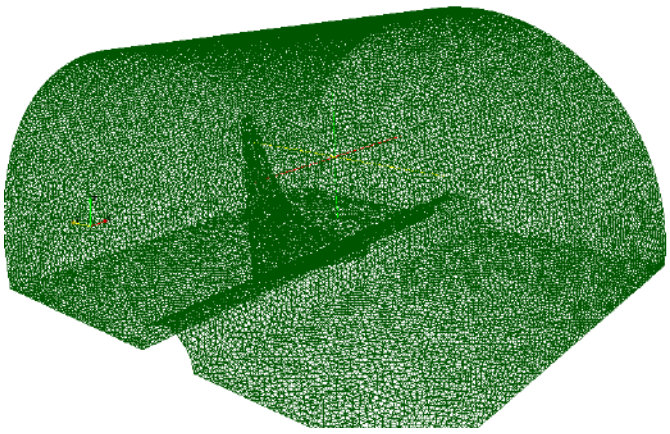
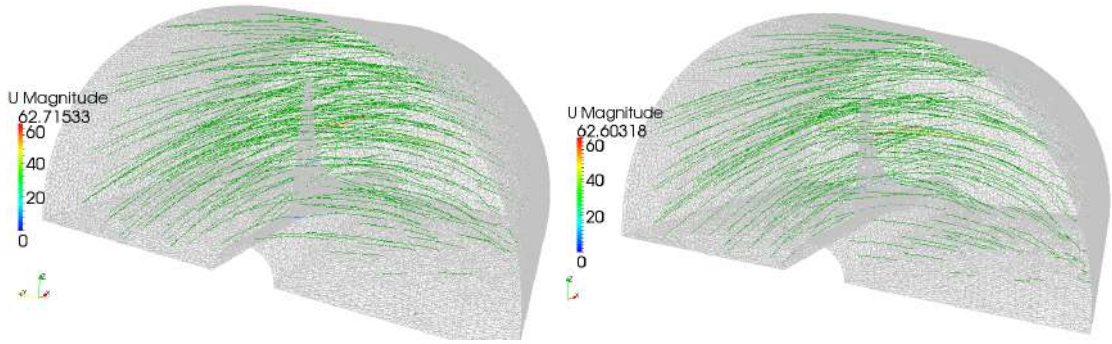


Figure 5.6: Mesh for a two-bladed wind turbine.



(a) Time step $t = 10^{-3}$ (b) Time step $t = 1$

Figure 5.7: Two-bladed turbine simulation: resulting stream lines.

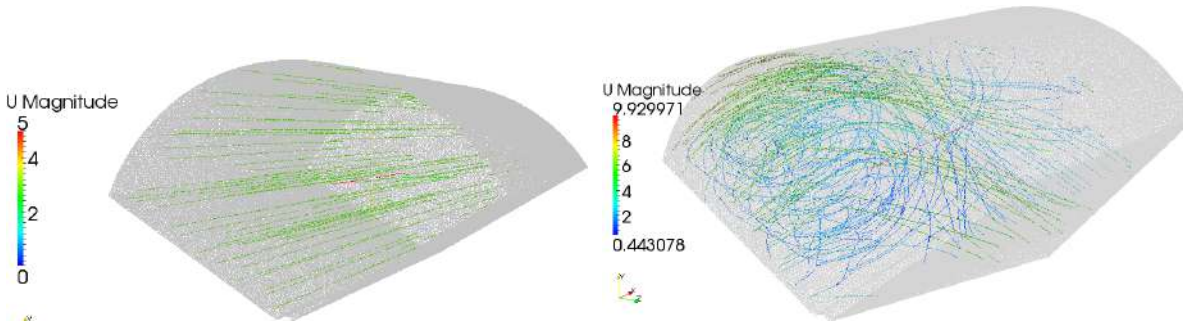
and the numbering on one of the elements to be connected is m, then the number of the correspondent must be $n + m$. In the case where the mesh was not defined in *blockMesh*, this is a problem as it is not intuitive. It had to be used the tool *createPatch* to connect the two periodic boundaries, however, the faces on the boundaries must completely coincident or this will not work, which is not easy when an unstructured mesh is being used. Using ICEMCFDs functions for defining periodic meshes and applying *createPatch*, it was finally possible to have a working periodic mesh in OpenFOAM.

In this case, as the shape of the blade was not yet fully defined, it was decided to use just the a mesh as simple as possible so that the results obtained could be extrapolated. The solution was to have a mesh with just the central spindle that goes all the way through the mesh.

Although this option seemed to be giving relatively good results with the purely axial flow inlet (Fig.5.8(a)), as soon as some more complicated situations were tested, as changing the mesh with the introduction of the blade or simply changing the inlet velocity to the cylindrical inlet, it would make the simulation explode near the cyclic boundaries, producing a flow like the one in Fig.5.8(b).

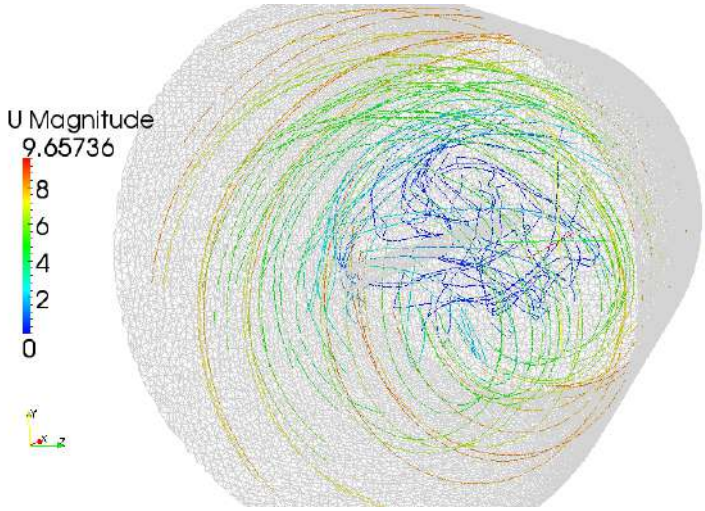
Abandoning the cyclic boundaries was then necessary to define a new mesh so the total domain had to be modeled. The geometry was kept the same as in the last case, and copied and rotated across the domain. This simulation also added the new definition of the inlet velocity where, instead of fixing the tangential speed, the value fixed was the angular velocity, as it would be in a wind turbine.

Finally, the results obtained were better (Fig.5.9), as the simulations converged, almost with the ve-

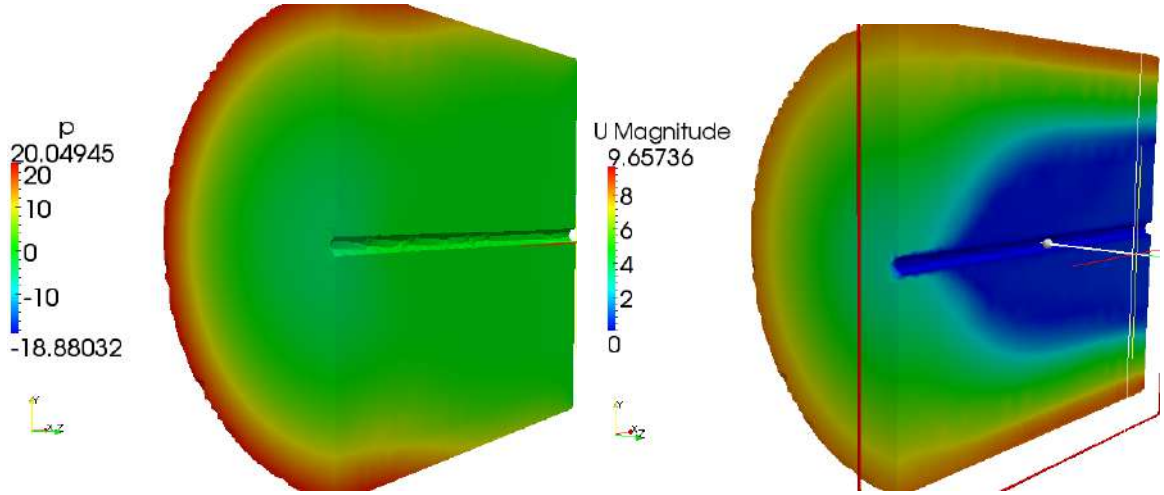


(a) simulation with the inlet purely axial (b) using cylindrical inlet conditions

Figure 5.8: Simulations using periodic boundary conditions.



(a) Stream lines



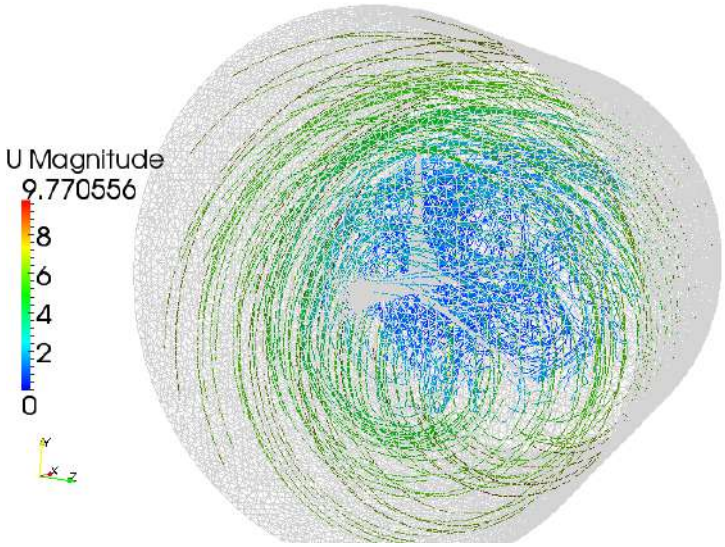
(b) Pressure (c) Velocity magnitude

Figure 5.9: Simulation of the entire domain with a central shaft.

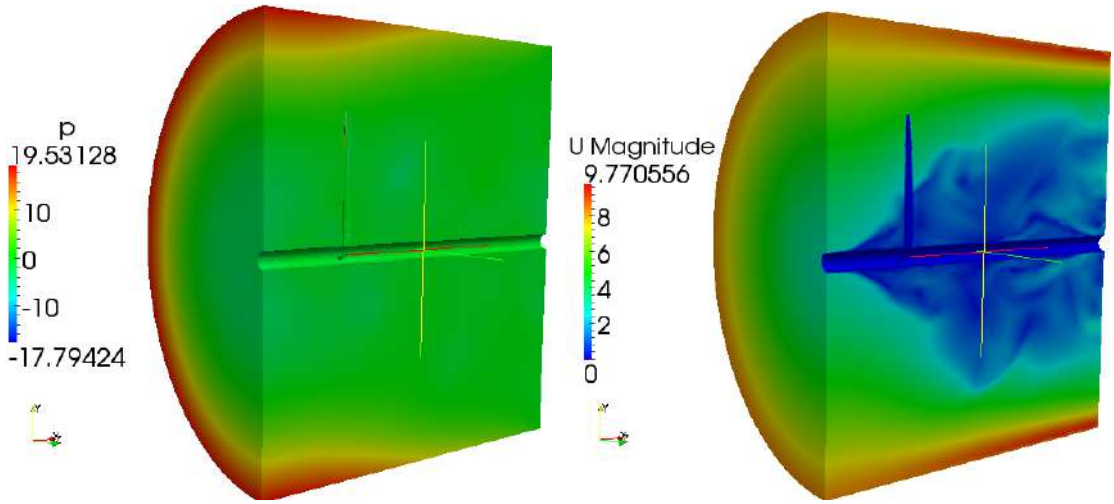
locity profile desired. The stream lines show that the flow generates a swirl around the central spindle. The results also showed a big wake, clearly visible both in the stream line layer and the velocity magnitude plot, this way considered a possible problem but as the simulation was steady it was decided to advance for the mesh with the blades, performing the exact same simulations.

5.1.3 Central Hub

As expected, the results obtained in Fig.5.10 were similar, where it is possible to observe that they are consistent with the ones previously obtained. However it also became clear that the shaft across all the domain is not a good option as the flow is reaching the blade already very disturbed, which was considered too unrealistic.



(a) Stream lines



(b) Pressure

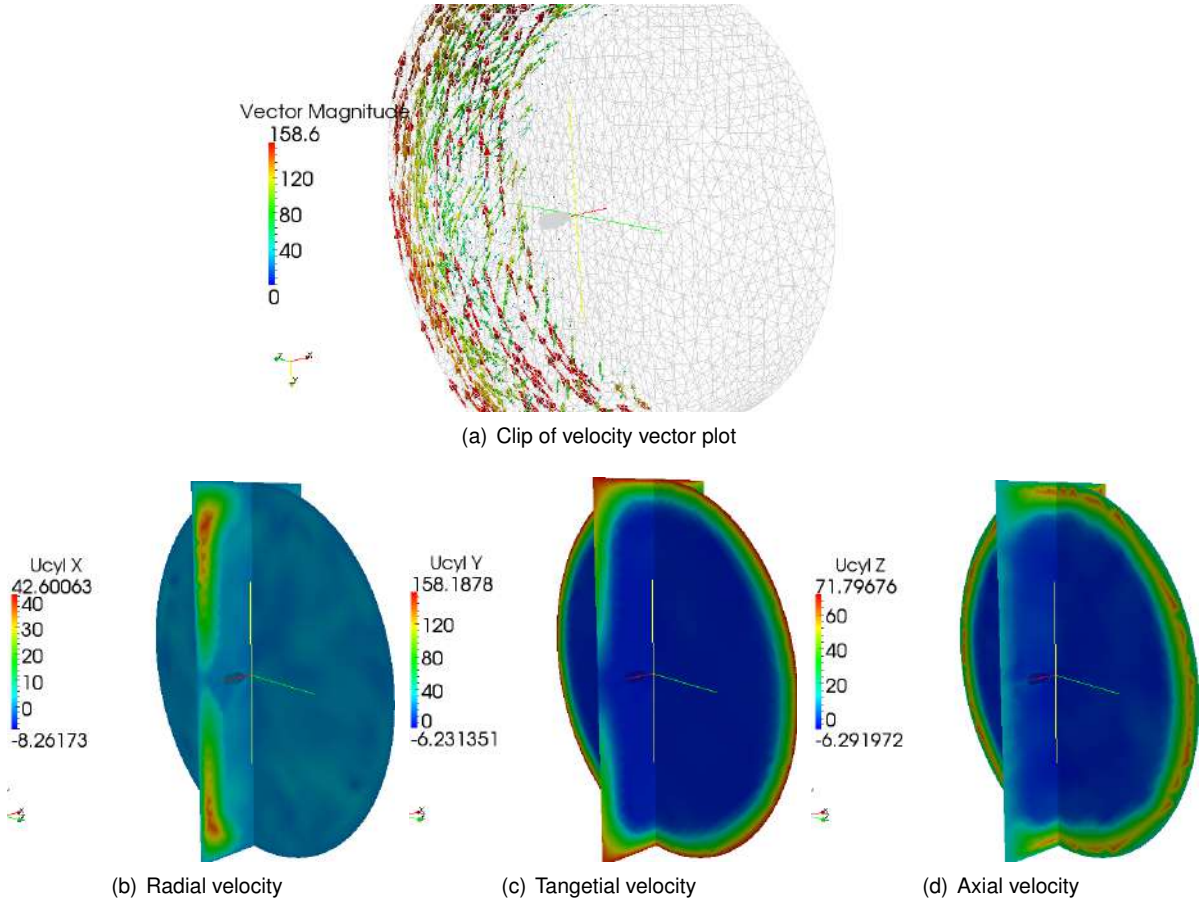
(c) Velocity magnitude

Figure 5.10: Simulation of the entire domain with a central shaft and blades.

These conclusions lead the work to a new test phase, where it was developed a central hub to attach the blades, with a similar format to the ones that are possible to find in current high power HAWT. It was defined with a cylindrical part in the center, and curved parts in the front and back. In addition, the domain was also made bigger, as it was considered that having the outer boundary as close as they were to the turbine blades might cause some problems.

The results presented in Fig. 5.11, show that indeed the problem with the wake emanating from the shaft is not a problem anymore, but a different problem arose: the boundary conditions are not

propagating as it was expected even when the simulation ran for a very long period and had reached convergence. The fluid near the hub is standing still and not moving as it was expected.



(a) Clip of velocity vector plot

(b) Radial velocity

(c) Tangetial velocity

(d) Axial velocity

Figure 5.11: Simulation of the entire domain with a central hub.

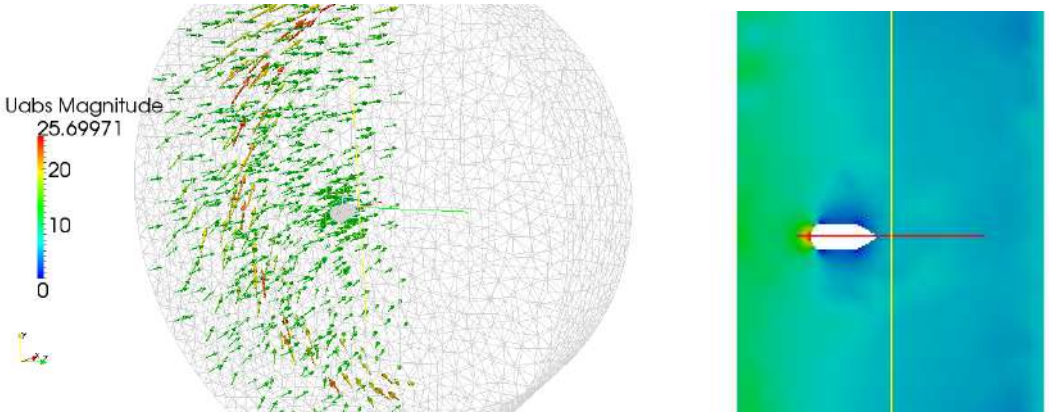
This simulation was followed by a large number of similar simulations where all the possible parameters were altered in order to try to increase the propagation in the flow. This turned out to be impossible, so it was once again necessary to find a different alternative.

5.1.4 Single Reference Frame

After it was settled that the simulations had to be completely different from what was done until this point, it is important notice that changing the solver in OpenFOAM requires a complete redefinition of the case and, as most of OpenFOAM advanced functions, it is not officially described anywhere how to define a case that would work. Having this in mind, the solver that could have the desired behavior was *simpleSRFFoam*, as it was close enough to the *simpleFoam* solver so the incompatibilities were expected to be small, and defining a single rotating frame does not require the mesh to have multiple zones defined, which is possible using an imported mesh from ICEMCFD but it was not possible to define one during the course of the current work.

Comparing the results presented in Fig.5.11 with the ones in Fig.5.12, the latter showed a much better behavior than the ones in the case using *simpleFoam*. The flow is now uniform, the stream

reached the hub with the conditions desired and pressure distribution also seemed consistent with a flow around the hub. In the pressure field, it is easy to identify the stagnation point on the front of the hub, and the quick decrease of pressure in the back.



(a) Clip of velocity vector plot (b) Pressure

Figure 5.12: SRF simulation mesh with central hub.

At this point, the domain, solver and boundary conditions necessary to model a HAWT rotor were properly defined so, it is possible to progress to the definition of the wind turbine flow simulation.

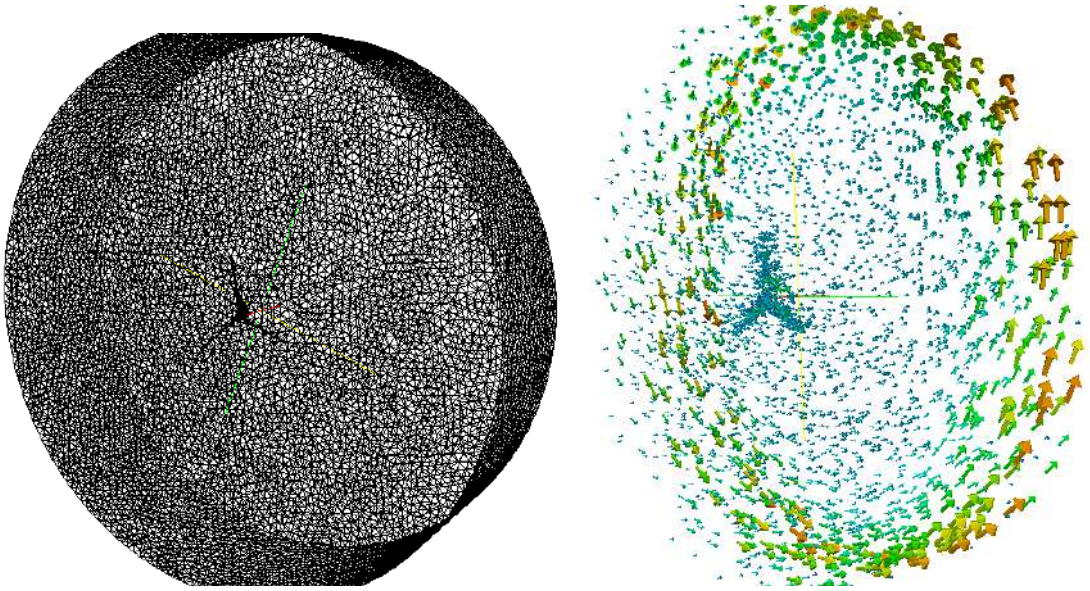
5.2 Wind Turbine Flow Simulation

Using *simpleSRFFoam* was then possible to simulate correctly the flow around the blades. The mesh used is presented in Fig.5.13, where the angular velocity was set to $-15.11rpm$ and the inlet axial velocity was set to $12.1m/s$. The results for the velocity vector field (Fig.5.13(b)), were as expected in a rotor of a wind turbine, and the convergence was achieved quickly and even the values for mechanic power produced, from the pressure differential between the side of the blades, seemed promising.

After defining the blades with the definition in Sec.3.1, the last concern was a way to connect them to the central hub. To give a more realistic feeling to the flow, the blades were joined to it with simple cylindrical connections. The hub and the connections are presented in gray in Fig.5.14. This is not a perfect solution as, in a real wind turbine, only a very small portion of the hub actually rotates, as the rest is used to store the electric generator, but this solution solved the problem explained before and it seemed to generate a wake much like it was expected. The dimensions and characteristics of the final simulation are sumarized in Tab.5.1.

Table 5.1: Final design parameters of a HAWT blade.

Rated wind Speed	$12.1m/s$
Diameter	$103.8m$
Number of Blades	3
λ_{design}	7.5
Rotor Speed	$15.11rpm$
Angle of Attack	2
Hub Diameter	$3m$
Hub length	$10m$



(a) Mesh (b) Velocity vector plot

Figure 5.13: SRF simulation of the rotor.

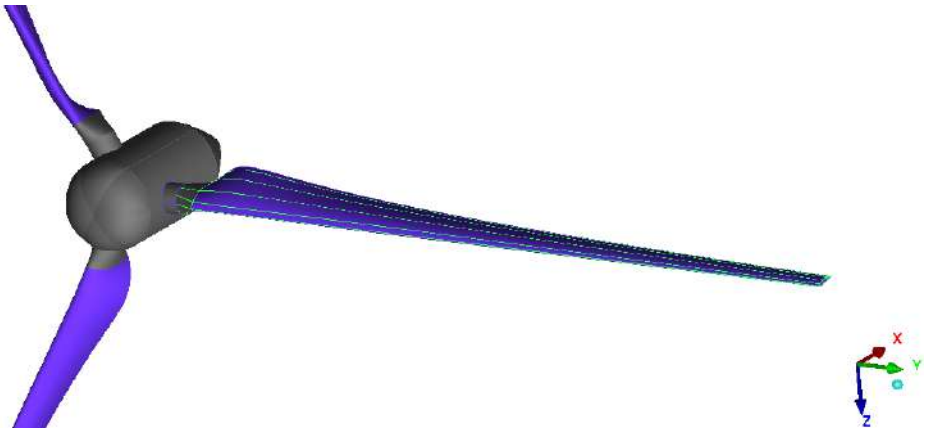


Figure 5.14: Final geometry of the HAWT rotor.

The outer surfaces of the mesh, where the inlet and outlet of the flow are located, were defined by a third of a cylinder defined using the dimensions of the blade as input parameters and then multiplying them to obtain a volume big enough so the outer boundaries did not affect the flow around the wall. It was first tried to run the simulations with only one sector of the rotor, using periodic boundaries, but it revealed to be unfruitful as the simulations had convergence problems. Using ICEMCFD geometry functions, the points, lines and surfaces created were copied and rotated to angles of 120^o and -120^o, forming the complete fluid domain presented in Fig.5.15.

The blocking on this geometry was done very carefully, to obtain a good discretization of the domain. This was a very time consuming process for a geometry with this level of complexity, but it was necessary when there was a non-fluid part totally inside the fluid, to differentiate what was fluid from what was void space, the empty space should not have a block. This geometry was then partitioned in 73 blocks, that defined all the fluid around the three blades and the hub.

The mesh generated was chosen to be tetrahedral, in order to model the blade with better detail,

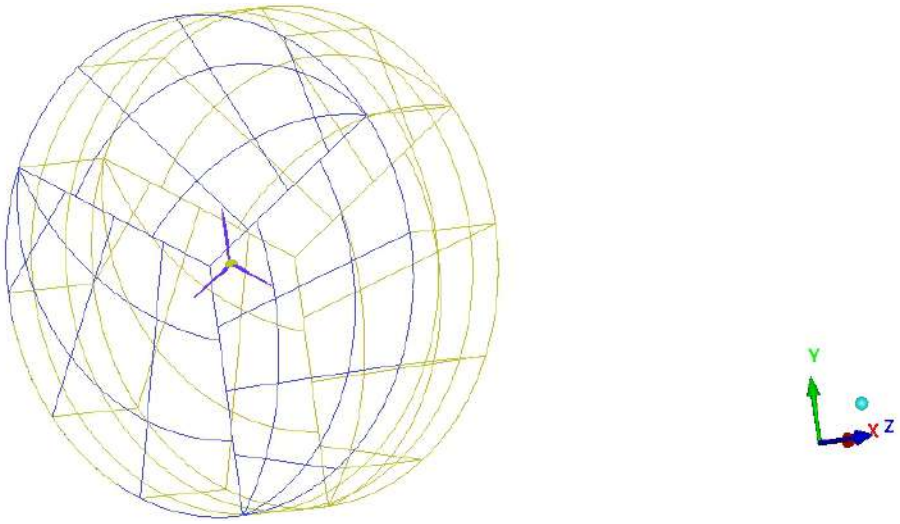


Figure 5.15: Fluid domain of the final fluid mesh.

and the element size was tunned such that it was possible to obtain good results without too much computational power. The rotor region was defined with more definition to account for boundary layer effects and obtain better results for the fluid structure coupling. The elements in this area are around five times smaller then the ones on the rest of the mesh. In an effort do maintain the simulation as quick as possible the mesh was restrained at a little above 1 million cells, which give it good definition. The final fluid mesh can be observed in detail in Fig.5.16.

This mesh sized corresponded to a computational time for iteration of about 9 s, while using simpleSRFFoam, which was considered very reasonable.

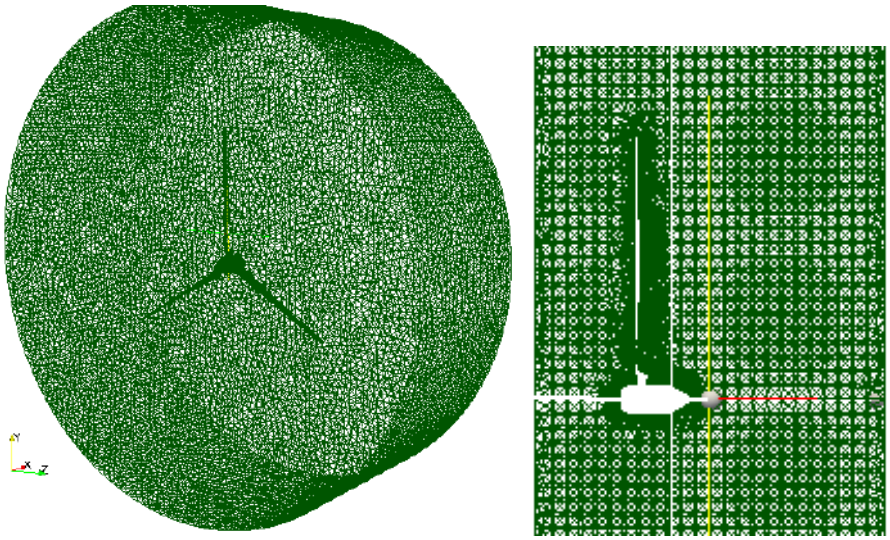
This mesh was also tested through a *simpleSRFFoam* simulation using the same conditions described above. The results for the simulation are presented in Fig.5.17, the velocity field is now exactly as it was expected, rotating at constant speed and with visible axial velocity, consistent with the previous cases.

If it is considered that the values are symmetric to the real ones due to the use of the rotating frame, the pressure on the rotor surface (Fig.5.18) also shows a distribution consistent with what is expected from a HAWT blade, with a clear pressure differential between the two sides, presenting a suction peak near the leading edge.

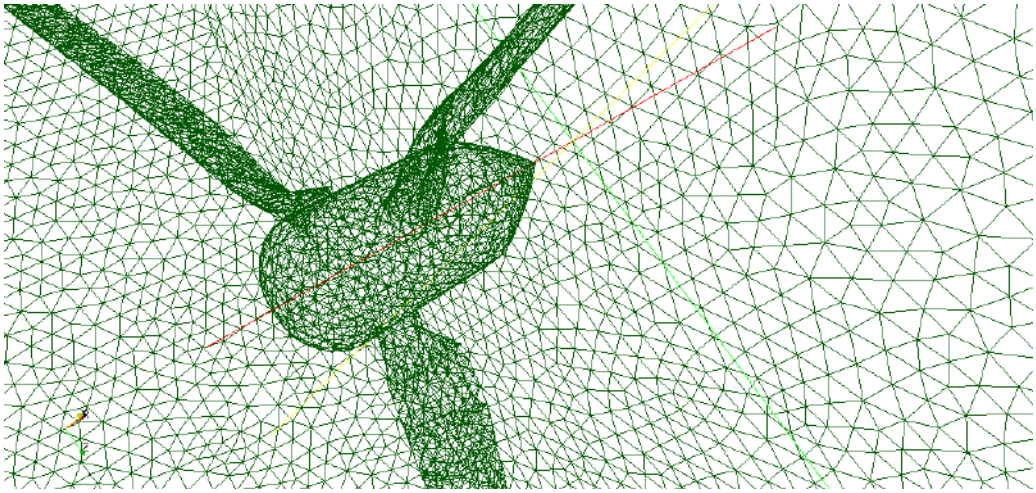
5.3 Solid Domain Definition

All structural analyses were performed using *stressedFoam*. As stated before, this is a very basic solver but it had to be tested in order to achieve the final step in this work. In this section, 3D cases were used to test the *stressedFoam* solver, creating the conditions to define the structural mesh for the blade.

The simulation presented in Fig.5.19, was one of the first tries with ICEMCFD meshes. As it possible to observe, applying a $60Pa$ stress on the tip of a steel beam ($E = 200GPa$, $\nu = 0.3$) did not have visible results since the displacements obtained were very small. On the other hand, it is possible to observe an irregular behavior in terms of equivalent stress. The results obtained in this field clearly point

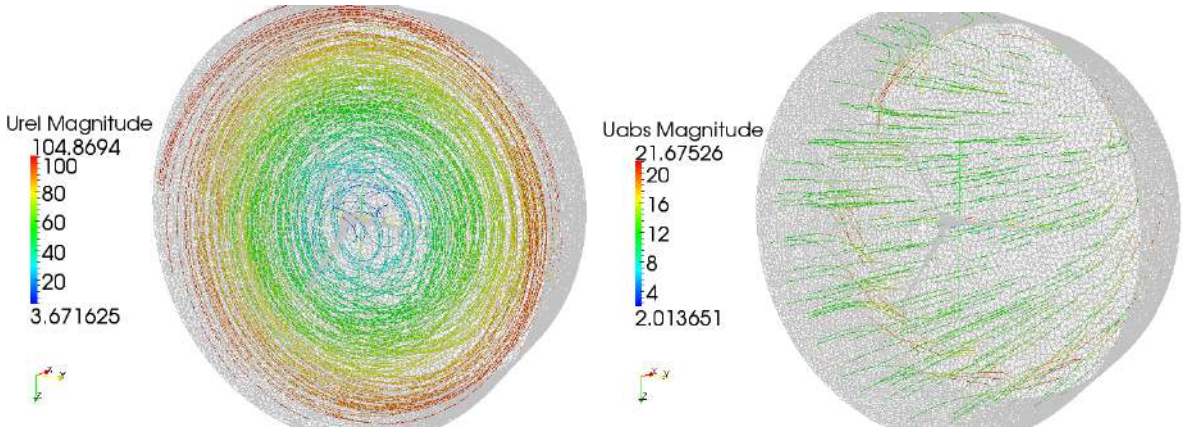


(a) Overview (b) Transversal cut



(c) Rotor Detail

Figure 5.16: Final fluid mesh details.



(a) Stream lines for the relative velocity (b) Stream lines for the absolute velocity

Figure 5.17: Final results for the flow simulations.

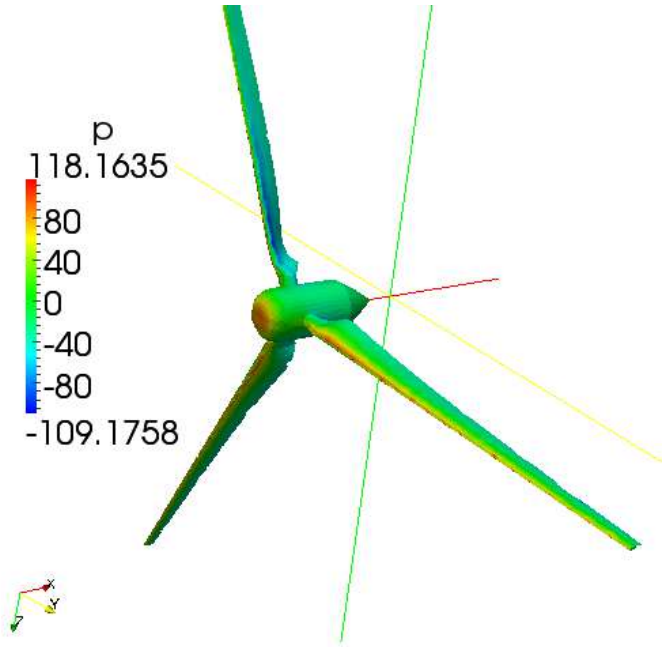
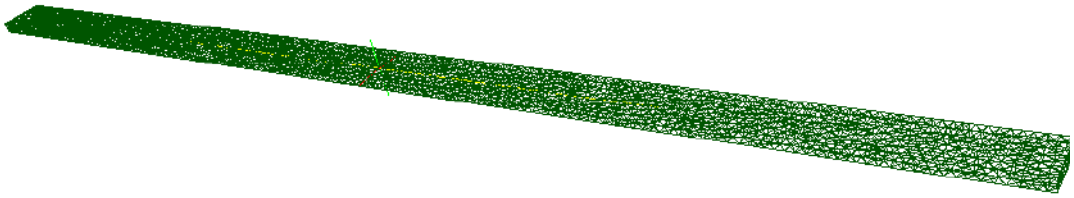
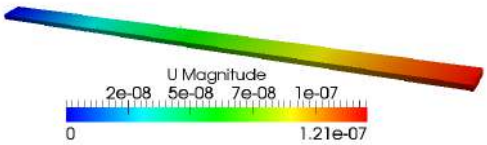


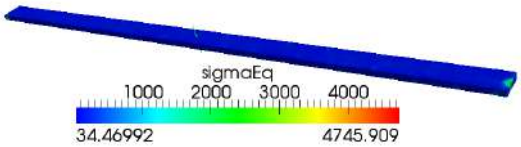
Figure 5.18: Pressure on the rotor.



(a) Mesh



(b) Displacement



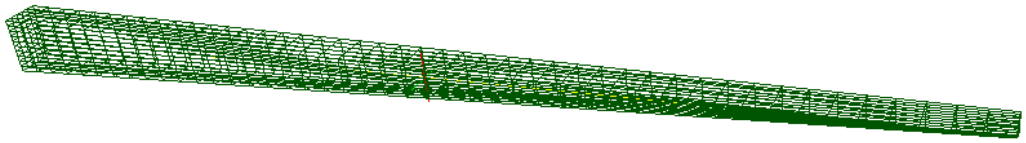
(c) Equivalent stress

Figure 5.19: *stressedFoam* test with rectangular beam.

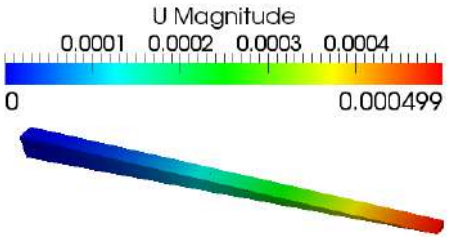
to numerical errors, due to the mesh. This type of errors were not only visible in this case, but also in other were the same mesh was used.

Trying to understand better the errors obtained the previous case and other similar errors that were obtain with more complex meshes, simulations using generic beam models created with *blockMesh* were made, all obtaining physically correct results. One of those is presented in Fig.5.20, where a tapered beam with a rectangular base and a triangular tip, is subjected to a bending force. This simulation was chosen as it is the one where the geometry is closer to the general format of the HAWT blade.

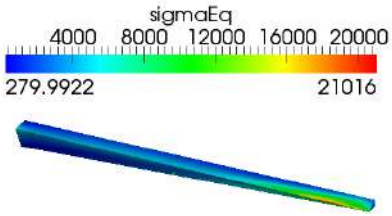
The results for displacements (Fig.5.20(b)) and equivalent stress (Fig.5.20(c)) seemed much more realistic than the ones presented in the previous case. The beam deformed as expected and most of the stress was concentrated in the thinner part of the mesh, as it should be.



(a) Mesh



(b) Displacement



(c) Equivalent stress

Figure 5.20: Tapered beam simulation using *stressedFoam*.

This simulations clarified that to used *stressedFoam*, it would be necessary to define a structured mesh as none of the tetrahedral meshes seemed to produce acceptable results.

5.4 Wind Turbine Blade Structural Simulation

Although it was one of the main points in the work, it was necessary to simplify the structural model due to time restriction. Using the same points generated from the code explained in the Sec.3.1, it was decided to eliminate the contributions of the leading and trailing edge of the blade in order to have a more regular mesh. As it seemed that the problems with the ICEMCFD meshes in *stressedFoam* were related to the unstructured meshes, it was created a structured mesh for the wind turbine blade that could represent this blade accurately using only hexahedral elements.

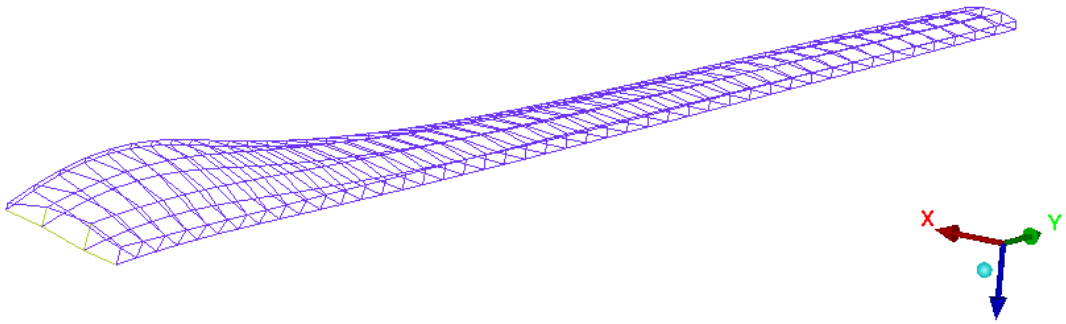


Figure 5.21: Blade structural mesh.

The blade is then defined as a solid with hexahedral elements seen in Fig.5.21, with similar structural characteristics to aluminum ($E = 70GPa$, $\nu = 0.3$). This is a very simplified solution but, as it was described by F.M. Jensen et al. [28], most of the structural loads of the blade are supported by webs in the center part of the blade, as so that this is the instrumented part on structural testings as is illustrated

in Fig.5.22 which shows a schematic from a instrumented wind turbine blade.

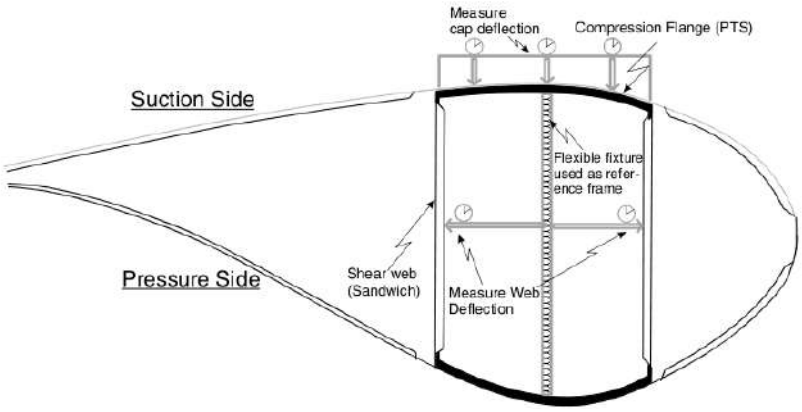
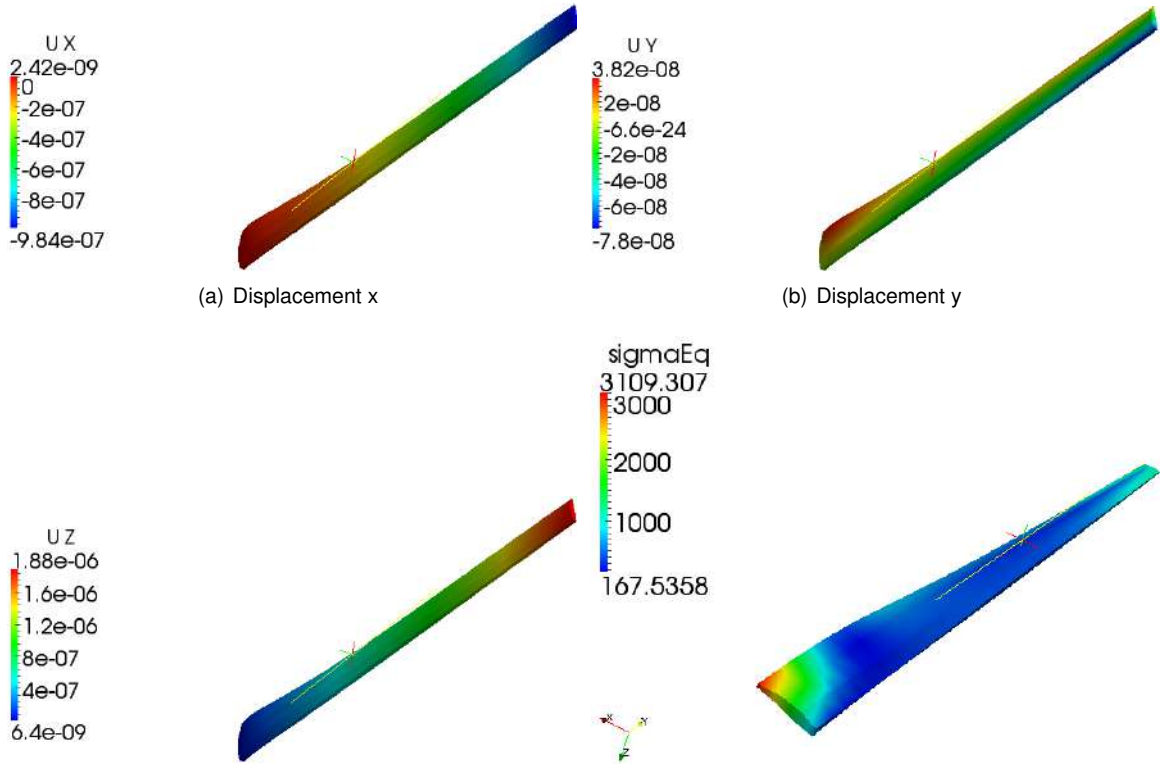


Figure 5.22: Local displacement instrumentation for structural test in a blade section [28].

Using this mesh, and applying a pressure of $600Pa$ in the z direction at the tip, the results obtain are presented in Fig.5.23. The displacements results were as expected, almost zero in the other directions and is only a valuable value in the direction where the force is applied.

As for the stress, the results obtained show, that the major concentration is in the thinner portion in the clamped top. These results were physically consistent and much better than the results shown in Fig.5.19(c), where the stress concentrations seemed to be in random places.



(a) Displacement x

(b) Displacement y

(c) Displacement z

(d) Equivalent stress

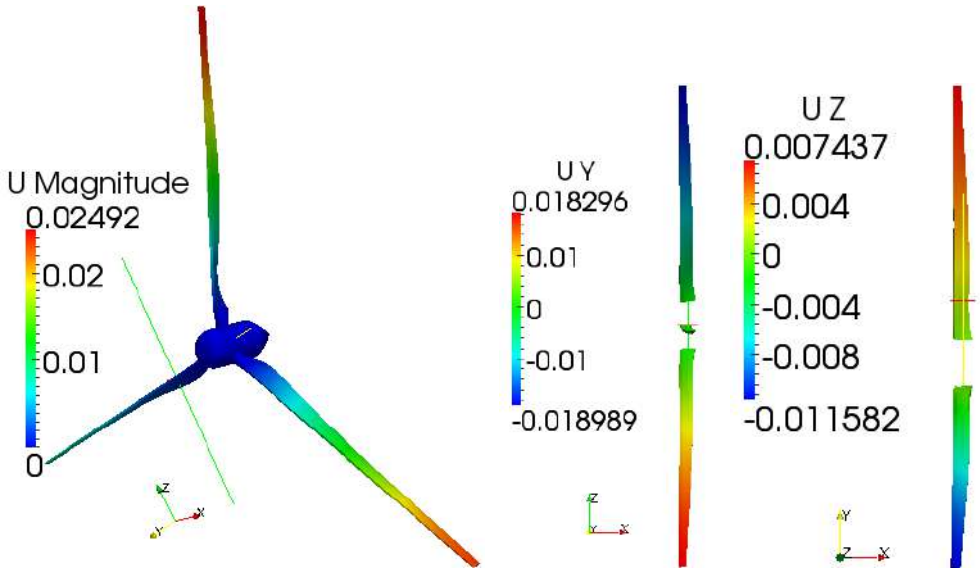
Figure 5.23: Simulation of the blade using *stressedFoam*.

As the results seemed good enough, this was the mesh that was used in the following FSI studies.

5.5 FSI Simulation and Results

After all the studies that had to be made to reach this point, the results for the fluid-structure interaction in a HAWT are presented in this section. The meshes, materials properties and fluid boundary conditions were the same that were defined in the last case for each part, with the necessary difference that now the blade was left free to deform in both solid and flow meshes.

The results obtained for the deformation are shown in Fig.5.24. These displacements, considering all the simplifications that were made, are impressive. If a solid metal beam dislocates more than $2cm$ just by the effects of the flow, it is possible to infer that this displacements, can indeed become a problem. Furthermore, it is also notable that the major displacements happen in the directions normal to the flow. This clearly shows that the solver is being capable os passing the single rotation frame results to the structural domain. Considering Fig.5.24(b) and Fig.5.24(c), it is also identifiable that the blades are moving acording to the flow, as in any of the directions, the blades are moving in oposite directions, denoting rotation.



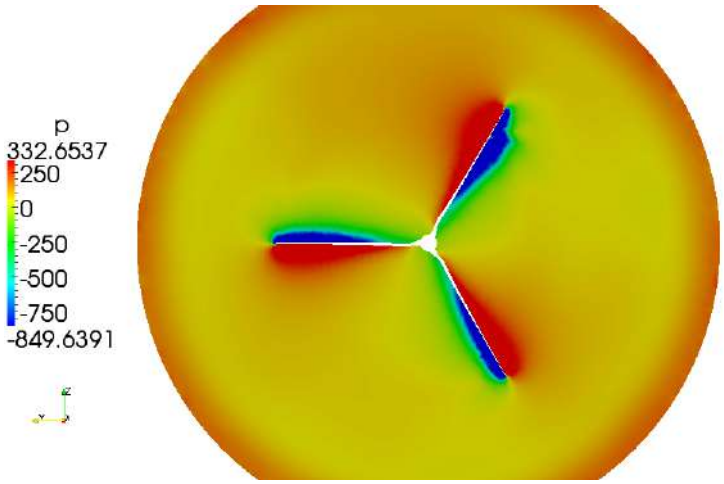
(a) magnitude (b) y direction (c) z direction

Figure 5.24: Displacement in the blades due to the flow.

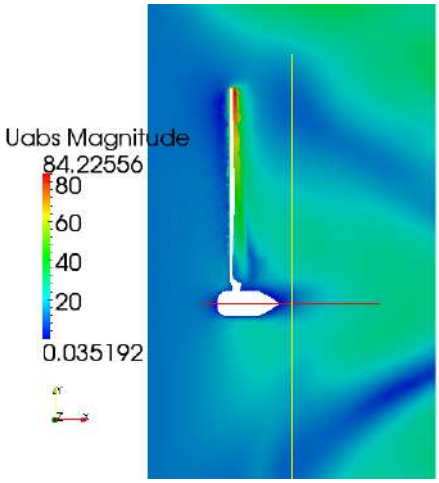
In the flow domain, the results shown are also interesting: as shown in Fig.5.25(a), the pressure differential between the sides of the blade is clear, pushing the blade due to aerodynamic forces, and producing power as shown next. The velocity profile also showed physically correct results, the domain is definitely moving at constant rotational speed as the velocity increases linearly, from the root of the blade to achieve the maximum value on the tip(Fig.5.25(b)).

It is also interesting to note that the computational time, despite using meshes which presented reasonable computational times individually, in the fsisimpleSRFFoam each iteration took about ten times the amount of time need for the equivalent fluid simulation, denoting the major disadvantage one of weakly coupled solvers.

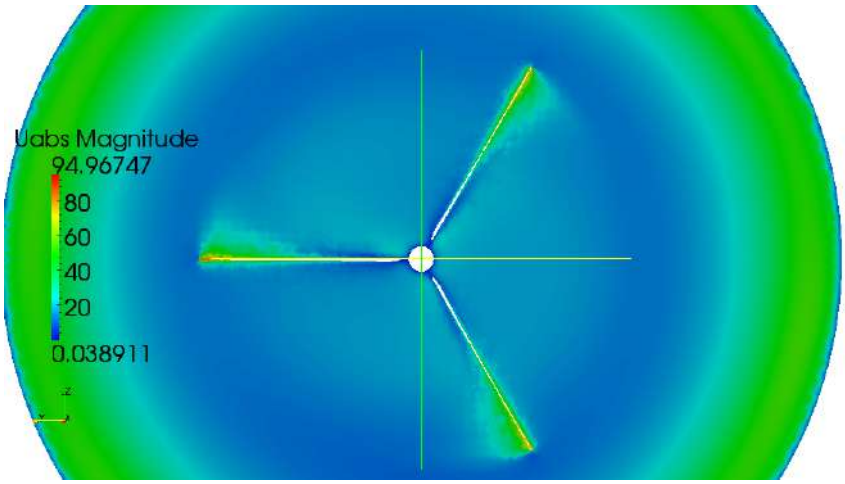
Using the script described in Sec.3.3, the values for the mechanical power and power coefficient



(a) Pressure



(b) Velocity magnitude (transversal cut)



(c) Velocity magnitude (axial cut)

Figure 5.25: Flow results on the FSI simulation of the HAWT blades.

were calculated and presented in Fig.5.26 and Fig.5.26. The results are probably a bit over predicted, since as the *forces* script in OpenFOAM might not be very accurate. Nevertheless, the turbine in which the parameterization was based [40] was projected to produce $5MW$ so the results obtained are in

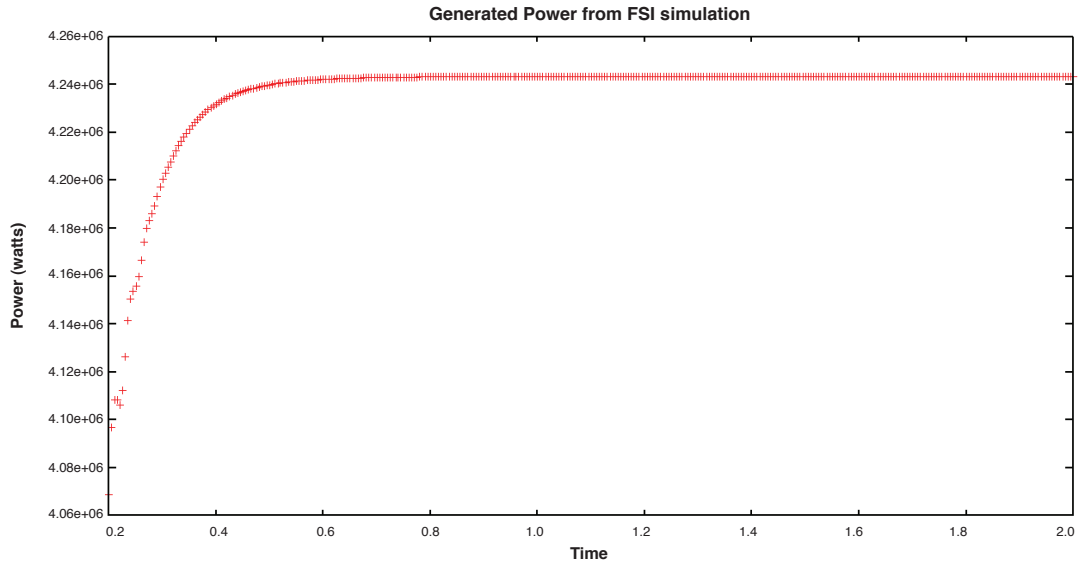


Figure 5.26: Power vs. simulation time.

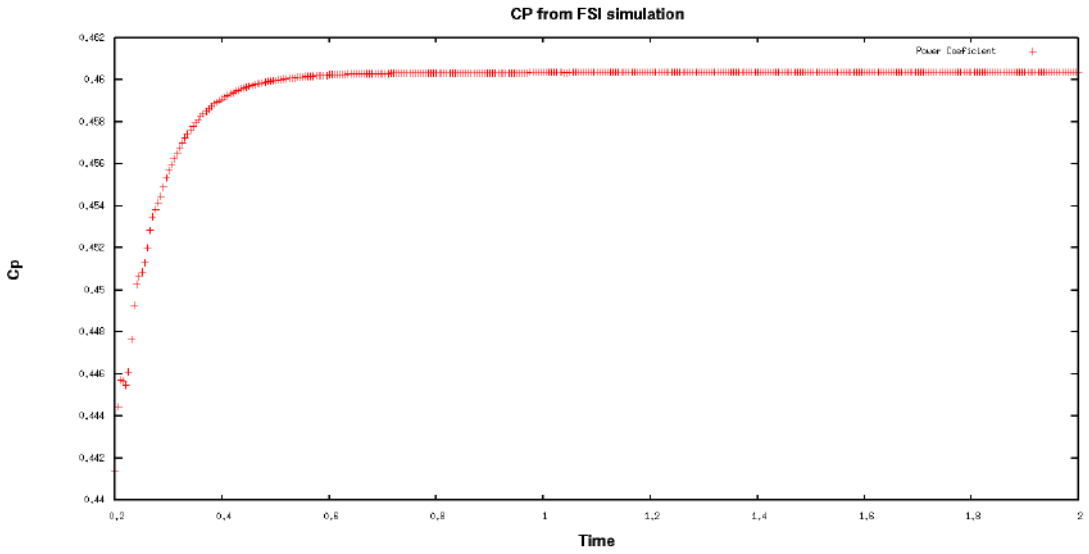


Figure 5.27: Power coefficient vs. simulation time.

concordance with this.

The values obtained of $4.24MW$ for produced power and C_p of 0.46 are consistent with current values for high power wind turbines but are probably too optimistic for a design in this early stage. Anyway, this demonstrates that the method for designing the blades for wind turbine works and produces acceptable results.

Considering the value obtained for thrust (Fig.5.28), the force in the direction of the axial flow, it is interesting to note that it is about one order of magnitude smaller then the forces produced on the blades, making it a smaller problem in a structural point of view.

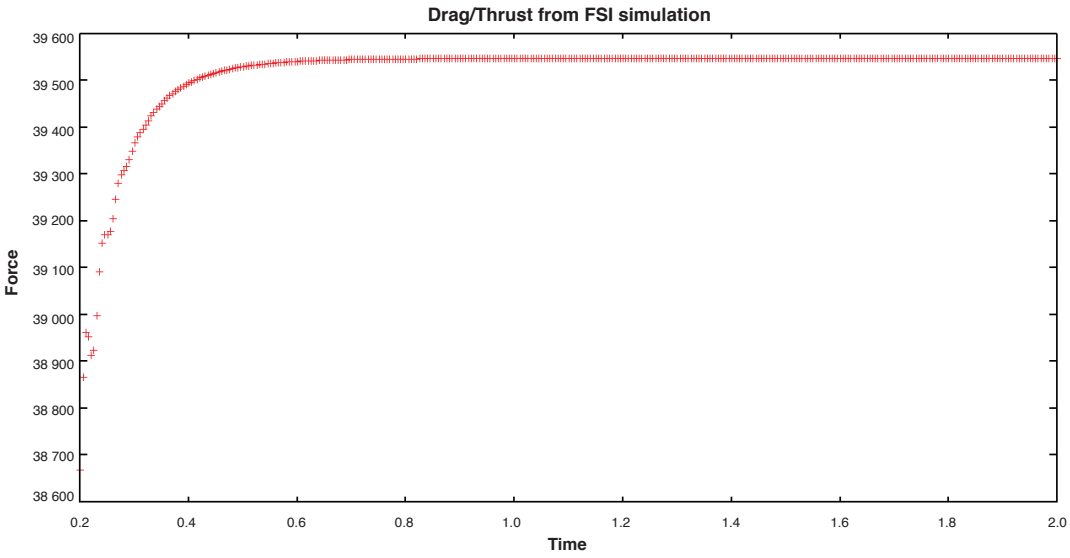


Figure 5.28: Trust vs. simulation time.

Chapter 6

Conclusions

This chapter comprises a general discussion over the achievements of the present work, wrapping it up with relevant conclusions. All topics described in the document are here revisited, complemented with the author's final thoughts on each subject and closing with a proposed follow up work-flow.

First of all, it is important to notice that, even if this was doubted several times during the course of this work, it is possible to accurately produce simulations of complex systems, such as a high power HAWT, using open-source software, namely OpenFOAM. With the appropriate base knowledge and after development of the needed tools, it is not much more complex than the commercial available options and it has several advantages over them.

OpenFOAM is currently at a stage were it is powerful enough to be useful for an extended analysis but, at the same time, it still has the simplicity needed for a community developed software, which makes it a perfect learning tool. The partitioned form of coding can be understood by engineering students, with proper guidance, being a great tool to learn basic implementation of finite volume solvers and to get more people interested in this type of work.

After the work that was done, it is safe to say that the most difficult part of using open-source software is the need to comprehend several programming languages and program specific commands which only become obvious after some time exploring the program. This becomes even more difficult when the objective, as happened in the current work, is to use and link several software to create a automatic process. As there is no official support for the OpenFOAM, using it without being able understand the source code is not very useful but, as previously mentioned, knowing some basics of $C++$, it becomes possible to understand the coding for each process after some time, making the definition of the cases much easier.

The choice to use the combination of a ANSYS ® product for the mesh generation and OpenFOAM for solving was probably not as good as it was supposed to, as ICEMCFD is a fairly new software and, because of that, it has still a small user base and it is even smaller if considering only the users which are exporting the meshes generated to OpenFOAM. This situation makes it much harder to find solutions that will work for complex problems, simply because there are no reports of someone trying that before.

It is now clear that taking advantage that *blockMesh*, OpenFOAM and ParaView are being developed to reach complete integration, conjugating all of these solutions would have been a better choice. As the results obtained when the mesh was created directly in blockMesh seemed to be much more accurate, the structured mesh helped to reach convergence much quicker and the simulations were more robust, while frequent crashes occurred with the meshes generated in ICEMCFD. As explained previously, learning *blockMesh* at the depth needed to generate the mesh for the complete HAWT rotor would not be feasible in a short term assignment, but it could be important for more advanced stage in the development.

Considering all the advantages and disadvantages, it is possible to conclude that OpenFOAM is a good tool for design and computationally test wind turbines. Although it has not been easy, and there is still some work to be done so that the design could be considered finalized, the final results were valid. One of the biggest problems in this work was the lack of experience in turbomachinery cases, which lead to the need to perform a lot of background test cases in order to understand the problems and their proper definitions. The use of reference frame solvers solved the problems of convergence and stability of the simulations; if it had been used earlier in the development of the project, it would have been possible to achieve a more refined design of the blade.

Apart from what was explored in this work, there are a lot more pre and post-processing tools made for OpenFOAM, which can become useful in further development of this process. It is possible to highlight pyFOAM, which is a python library dedicated to the automation of OpenFOAM cases. This library has functions to change conditions in an automatic way, define new cases from a selected case and also, reading the log files and producing helpful data, as residual charts.

Fluid-structure coupled design is one of the areas which should have a major boost in the next few years, as considering these effects from an early stage in the design process can make serious improvements in the final design, and the applications where this coupling is indeed important are countless. For example, highly flexible components are now preferred in all solutions that require hinges and other mechanical components and, in these components, it is essential to have a comprehensive FSI study.

Regarding the development of the FSI solver, although it is considered successful reusing parts of code from standard solvers, it is not perfect because OpenFOAM is missing a good structural solver that could make the FSI solver for the analysis of wind turbine blades much more accurate.

The results obtained are a clear indicator that this is a problematic area in the design of wind turbine blades. Even with all the simplifications made, ending up with a structural model that is not very realistic, the results can be extrapolated to a more realistic scenario. The results show that if the blade was a solid aluminum beam, the maximum displacement caused by flow induced forces would be around two centimeters, which means that if the blade was more realistic, that is, if the structure was much more flexible than the one that was tested, the results could have been really impressive.

The pressure and velocity profiles were not altered too much when changing for the FSI solver but, if the results for the 2D case that was used to validate the results of the solver are considered, it is possible to infer that if the blade was modeled following a blade structure, there would be significant the changes in velocity and pressure profiles.

As discussed in the previous chapter, the predictions for power output seemed very good, almost too good to be believable. These predictions also show that even though the pressure and velocity might not have converged so quickly, integral proprieties, such as power and ”thrust”, seemed to converge much faster, once again showing that the simulations were stable.

As all the process was planed to be automated, it will not be too difficult to implement simple optimization techniques to improve design of this blade. Using a simple gradient-based method would be a good start to optimize something small like the optimum angle of attack or the thickness of the load bearing spar, which even though are important parts of the design did not get as much focus as they as they should have.

For a more advanced optimization process, as there are not many input parameters, and mainly the output should be focused on the produced power, while not forgetting to have extras condition to make sure that the blades are structurally viable, it is considered that genetic optimization might be the correct approach to further enhance this design. As this algorithm is blind to what is being simulated, it can produce multidisciplinary designs that take into account all the aspects being studied, without having to decide which discipline to favor, getting the overall best design according to specifications.

To finalize this work it is important to acknowledge that wind energy production, is still far from reaching its full potential but, in any case, every big advance in the technological world is made from an enormous amount of small steps, like the one taken with the work developed here. Multidisciplinary studies have always been an important part of engineering designs but it will only in the future reach a point that for every component, there should be a parallel study, developing the best possible design that accounts for every performance limitation.

6.1 Main Achievements

The main achievements of the present work can be summarized as follows:

- development of FSI solvers for OpenFOAM, which can be useful for a series of applications requiring turbulent flow. These solvers can also be useful as a basis for more complex weakly coupled solvers to be developed in the future;
- FSI simulation of a wind turbine rotor using only open-source software and the specially developed solver, which was one of the main objectives and was not found a similar case in the literature review.

6.2 Future Work

With the completion of this work there are a lot of options to expand and improve the development made until now. The most urgent topic that needs to be addressed is the mesh for the structural solver, as it is important to have a mesh that can accurately describe what is being modeled. The base design should be much like a airplane wing, with a structural foundation with spars and ribs that go all the way through

the blade, forming a boxed structure, and then completed with a composite skin to give the aerodynamic shape and distribute the flow induced forces.

To have a FSI solver that is useful for designing wind turbine blades, it is necessary that it is able to solve structural problems using composite materials. Without it, the blade will never be modeled correctly as there are not any current blade designs that do not use composite materials as structural elements and skins. This is the major limitation of *stressedFoam*, in which, it is only possible to analyze isotropic materials. In the current state of the solver it also is not possible to acquire stress states, which also should be a top priority.

Considering the flow solver, there is also room for improvements using the multi reference frame technique. This approach was not pursued due to the computational drawback of not having a dedicated machine to perform the simulations. It is pointed out in some technical literature that using this technique to define a very large domain with quiescent air and having a smaller domain rotating in the center, using rotating interfaces between the two, should produce better results than the single reference frame, but this would exponentially increase the simulation time and it was simply not feasible in a personal computer.

It would also be interesting and important to make optimization studies of this blade design. For instance, improving the shape of the airfoils, which does not require any significant modification of the script developed since the files for the airfoil data are given as inputs and can be altered at any time. It could also be a good option to optimize the chord and twist distributions, as the parameterization used only took into account aerodynamic optimization.

Finally, it could be an interesting idea to use the FSI solver and couple it to other disciplines as well, like aeroacoustic and fatigue analysis, having like this a true multidisciplinary design and optimization.

Globally, it is considered that this work was successful, pointing out some important questions and creating a large enough base to develop the subject of multidisciplinary wind turbine blade design further.

Bibliography

[1] V. Casas, F. Pena, A. Lamas, and R. Duro, "An Evolutionary Environment for Wind Turbine Blade Design," in *Computational Intelligence and Bioinspired Systems*, ser. Lecture Notes in Computer Science, J. Cabestany, A. Prieto, and F. Sandoval, Eds. Springer Berlin / Heidelberg, 2005, vol. 3512, pp. 812–818, 10.1007/11494669_146.

[2] R. Energy and A. U. P. (U.S.), *Technology white paper on wind energy potential on the U.S. Outer Continental Shelf (electronic resource)*. U.S. Dept. of the Interior, Minerals Management Service, Renewable Energy and Alternate Use Program, Washington, D.C., 2006.

[3] M. Sathyajith, *Wind Energy: Fundamentals, Resource Analysis and Economics*. Springer, 2006.

[4] T. Burton, *Wind Energy Handbook*. J. Wiley, 2001.

[5] A. Betz, *Introduction to the Theory of Flow Machines. (D. G. Randall, Trans.)*. Oxford: Pergamon Press, 1919/1966.

[6] G. J. Herbert, S. Iniyan, E. Sreevalsan, and S. Rajapandian, "A review of wind energy technologies," *Renewable and Sustainable Energy Reviews*, vol. 11, no. 6, pp. 1117–1145, 2007.

[7] J. Park, J. Kim, Y. Shin, J. Lee, and J. Park, "3MW class offshore wind turbine development," *Current Applied Physics*, vol. 10, no. 2, Supplement, pp. S307–S310, 2010, the Proceeding of the International Renewable Energy Conference and Exhibition 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1567173909005288

[8] P. S. Veers, T. D. Ashwill, H. J. Sutherland, D. L. Laird, D. W. Lobitz, D. A. Griffin, J. F. Mandell, W. D. Musial, K. Jackson, M. Zuteck, A. Miravete, S. W. Tsai, and J. L. Richmond, "Trends in the design, manufacture and evaluation of wind turbine blades," *Wind Energy*, vol. 6, no. 3, pp. 245–259, 2003.

[9] T. D. Ashwill, "Materials and Innovations for Large Blade Strucutures: Research Opportinities in Wind Energy Technology," in *Proceedings of the* 50^{th} *AIAA Structures, Structural Dynamics & Materials Conference*, May 2009.

[10] J. Buist and D. Milborrow, "Europe 2020 Wind Energy Targets," European Wind Energy Association (EWEA), Wind Power Montly, Special Report, 2011.

[11] E. W. E. Association, "Wind Energy - the facts, Part I - Technology," http://www.wind-energy-the-facts.org/documents/download/Chapter1.pdf, 2009.

[12] “Technology White Paper on Wind Energy Potential on the U.S. Outer Continental Shelf,” Accessible from http://ocsenergy.anl.gov, May 2006.

[13] “Energia Eolica em Portugal 2011,” REN - Rede Electrica Nacional. Available at www.ren.pt, 2012.

[14] L. Greco, C. Testa, and F. Salvatore, “Design Oriented Aerodynamic Modelling of Wind Turbine Performance,” *Journal of Physics Conference Series*, vol. 75, no. 1, p. 012011, 2007.

[15] W. A. Timmer and R. P. J. O. M. van Rooij, “Summary of the Delft University Wind Turbine Dedicated Airfoils,” *ASME Conference Proceedings*, vol. 2003, no. 75944, pp. 11–21, 2003. [Online]. Available: http://link.aip.org/link/abstract/ASMECP/v2003/i75944/p11/s1

[16] R. P. J. O. M. Van Rooij and W. A. Timmer, “Roughness sensitivity considerations for thick rotor blade airfoils,” *Journal of Solar Energy Engineering*, vol. 125, no. 4, p. 468, 2003.

[17] H. Glauert, *Windmills and fans - Aerodynamic theory*. Julius Springer, 1935.

[18] M. Hansen, J. Sørensen, S. Voutsinas, N. Sørensen, and H. Madsen, “State of the art in wind turbine aerodynamics and aeroelasticity,” *Progress in Aerospace Sciences*, vol. 42, no. 4, pp. 285 – 330, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0376042106000649

[19] S. C. N. N. Sørensen, J. Johansen, “CFD Computations of Wind Turbine Blade Loads During Stand-still Operation, Technical Report,” Risø National Laboratory, Roskilde, Denmark, Tech. Rep., 2004.

[20] D. A. Digraskar, “Simulations of flow over wind turbines,” Master’s thesis, University of Massachusetts Amherst, May 2010, mechanical and Industrial Engineering.

[21] OpenCFD, “OpenFOAM - The Open Source CFD Toolbox - User’s Guide,” OpenCFD Ltd. version 1.6, http://www.openfoam.org/archive/1.6/docs/user/, April 2009.

[22] D. Hartwanger and D. A. Horvat, “3D modelling of a wind turbine using CFD,” *NAFEMS Conference*, 2008.

[23] N. Sezer-Uzol and L. N. Long, “3-D Time-Accurate CFD Simulations of Wind Turbine Rotor Flow Fields,” *44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, 9-12 January*, 2006.

[24] H. Nilsson, “Evaluation of OpenFoam for CFD of turbulent flow in water turbines,” in *Proceedings of the* 23^{rd} *IAHR Symposium - Yokohama*, Oct 2006.

[25] M. A. Sprague, P. J. Moriarty, M. J. Churchfield, K. Gruchalla, S. Lee, J. K. Lundquist, J. Michalakes, and A. Purkayastha, “Computational modeling of wind-plant aerodynamics,” *NREL - National Renewable Energy Laboratory*, 2010.

[26] T.Kinsey and G. Dumas, “Parametric study of an oscilating airfoil in a power-extraction regime,” *AIAA Journal*, vol. 46, no. 6, pp. 1318–1328, June 2008.

[27] “OMNIDEA, HAWE High Altitude Wind Energy,” http://www.omnidea.net/hawe/, 2012.

[28] F. Jensen, B. Falzon, J. Ankersen, and H. Stang, "Structural testing and numerical simulation of a $34m$ composite wind turbine blade," *Composite Structures*, vol. 76, pp. 52 – 61, 2006, iCCM-15 Fifteenth International Conference on Composite Materials.

[29] K. Y. Maalawi and H. M. Negm, "Optimal frequency design of wind turbine blades," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 90, no. 8, pp. 961 – 986, 2002.

[30] M. Bechly and P. Clausen, "Structural design of a composite wind turbine blade using finite element analysis," *Computers & Structures*, vol. 63, no. 3, pp. 639 – 646, 1997.

[31] D. J. Malcolm and D. L. Laird, "Modeling of blades as equivalent beams for aeroelastic analysis," *ASME Conference Proceedings*, vol. 2003, no. 75944, pp. 293–303, 2003.

[32] H. J. Bungartz and M. Schäfer, *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, ser. Lecture Notes in Computational Science And Engineering. Springer-Verlag, 2006.

[33] R. L. Campbell, "Fluid-Structure Interaction and Inverse Design Simulations for Flexible Turbomachinery," Ph.D. dissertation, The Pennsylvania State University, The Graduate School, College of Engineering, 2010.

[34] J. Lorentzon, "Fluid-structure interaction (FSI) case study of a cantilever using OpenFOAM and DEAL.II with application to VIV," Master's thesis, Lunds Institute of Technology, Sweden, June 2009, technical Mathematics.

[35] G. J. Kennedy and J. R. R. A. Martins, "Parallel solution methods for aerostructural analysis and design optimization," 13^{th} *AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, no. 9308, 2010.

[36] P. Fuglsang and H. Madsen, "Optimization method for wind turbine rotors," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 80, no. 1, 2, pp. 191 – 206, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167610598001913

[37] J. Bonnans, *Numerical Optimization: Theoretical and Practical Aspects*, ser. Universitext (1979). Springer, 2003.

[38] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, ser. Artificial Intelligence. Addison-Wesley, 1989.

[39] J. Mendez and D. Greiner, "Wind blade chord and twist angle optimization by using genetic algorithms," Institute of Intelligent Systems and Numerical Applications in Engineering, Univ. Las Palmas Gran Canaria, 2010.

[40] B. Kim, W. Kim, S. Bae, J. Park, and M. Kim, "Aerodynamic design and performance analysis of multi-MW class wind turbine blade," *Journal of Mechanical Science and Technology*, vol. 25, pp. 1995–2002, 2011, 10.1007/s12206-011-0521-x. [Online]. Available: http://dx.doi.org/10.1007/s12206-011-0521-x

[41] “Python Reference Manual,” PythonLabs, Virginia, USA, 2001. Available at www.python.org.

[42] “Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,” http://www.geuz.org/gmsh/.

[43] *ParaView*, 3rd ed., http://www.paraview.org/, Apr 2012.

[44] *The OpenFOAM®Extend Project*, 1st ed., http://www.extend-project.de/, 2012.

[45] F. P. K¨arrholm, “Numerical modelling of diesel spray injection, turbulence interaction and combustion,” Ph.D. dissertation, Department of Applied Mechanics, Chalmers University of Technology, G¨oteborg, Sweden, 2008, appendix.

[46] S. V. Patankar and D. Spalding, “A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows,” *Int. J. of Heat and Mass Transfer*, vol. Volume 15, no. Issue 10, pp. 1787–1806, October 1972.

[47] *Autodesk® Algor® Simulation User’s Guide*, build 15 ed., Available at http://download.autodesk.com/us/algor/userguides/mergedProjects/setting_up_the_analysis/fluid_flow/loads_and_constraints/rotating_frame_of_reference.htm, Apr 2009.

[48] H. Jasak and H. G. Wellerv, “Application of the finite volume method and unstructured meshes to linear elasticity,” *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 267–287, Feb 2000.

[49] K. J. Maus, “Constructing solvers for weakly coupled FSI problems using OpenFOAM-1.5-dev,” Norwegian University of Life Sciences, Tech. Rep., 2009.

[50] B. E. Launder and B. I. Sharma, “Application of the energy dissipation model of turbulence to the calculation of flow near a spinning disc,” *Letters in Heat and Mass Transfer*, vol. 1, no. 2, pp. 131–138, 1974.

[51] P. R. Spalart and S. R. Allmaras, “A one-equation turbulence model for aerodynamic flows,” *Recherche Aerospatiale*, no. 1, pp. 5–21, 1994.

Appendix A

Python Scripts

A.1 Parameterization of the Wind Turbine Blade

Example of one of the scripts for the parameterization of the blade.

```
import os
import string
from math import *
from numpy import *
from scipy import interpolate
import scipy as Sci
import scipy.linalg


#### function to export the points in ICEMCFD format
def write_points( filename, x , y, z, w, npontos = None, nlinhas = None):

        if w == 0:
                    fout=open( filename, 'w' )
                fout.write( '%d %d \n' % (npontos,nlinhas))
         else:
                    fout=open( filename, 'a' )
                fout.write( '%f %f %f\n' % (x,y,z))

        fout.close()
        return 1
##=====================================================

#### function to change the chord on the airfoil
def aply_chord(chord, x, y):
        xx =[]
```

```
        yy =[]
        for i in range( 0, len(x)):
                xx.append(chord*float(x[i]))
                yy.append(chord*float(y[i]))
        return xx,yy
##=======================================================

#### function to read airfoil data
def read_airfoil(filename, basex=None):

        fairfoil=open("/home/brunotojo/py/pyFoil/"+filename, "r" )
        fairfoil_content=fairfoil.readlines()
        edg_p=int(float(fairfoil_content[0].rpartition(' ')[-1]))


        x = []
        y = []


        for i in range( 1, edg_p+1 ):
                x.append(float(fairfoil_content[i].rpartition(' ')[0]))
                y.append(float(fairfoil_content[i].rpartition(' ')[-1]))

        coordinatex=array(x)
        coordinatey=array(y)

        xmax=coordinatex.max()
        lx=len(coordinatex)

        if xmax!=1:
                coordinatex=coordinatex/xmax
                coordinatey=coordinatey/xmax

        if basex!=None:
                x1=[]
                y1=[]
                x2=[]
                y2=[]
                bx1=[]
                bx2=[]
                i=0
                while coordinatex[i]!=0.0:
```

```
        x1.append(coordinatex[i])
        y1.append(coordinatey[i])
        i+=1
for j in range(i,lx):
        x2.append(coordinatex[j])
        y2.append(coordinatey[j])
i=0
while basex[i]!=0.0:
        bx1.append(basex[i])
        i+=1
for j in range(i,len(basex)):
        bx2.append(basex[j])

x1=array(x1)
y1=array(y1)
x2=array(x2)
y2=array(y2)
bx1=array(bx1)
bx2=array(bx2)

if x1[0]>x1[-1]:
        xaux=zeros((len(x1)))
        yaux=zeros((len(x1)))
        for i in range(0, len(x1)):
                xaux[i]=x1[len(x1)-1-i]
                yaux[i]=y1[len(x1)-1-i]
        x1=xaux.copy()
        y1=yaux.copy()

if x2[0]>x2[-1]:
        xaux=zeros((len(x2)))
        yaux=zeros((len(x2)))
        for i in range(0, len(x2)):
                xaux[i]=x2[len(x2)-1-i]
                yaux[i]=y2[len(x2)-1-i]
        x2=xaux.copy()
        y2=yaux.copy()

tck = interpolate.splrep(x1,y1)
 yy1 = interpolate.splev(bx1, tck)
tck = interpolate.splrep(x2, y2)
 yy2 = interpolate.splev(bx2, tck)
```

```
                coordinatex=array(basex)
                coordinatey=array(concatenate((yy1,yy2),1))

        return coordinatex, coordinatey

##=============================================================
#### function to calculate airfoil along the blade
def recalc_airfoil(coordinatex1, coordinatey1, coordinatex2, coordinatey2,
coordinatex3, coordinatey3,coordinatex4, coordinatey4,coordinatex5
, coordinatey5, ss1, ss2, ss3, ss4, ss5, mu):
        if mu==ss1:
                coordinatex=coordinatex1
                coordinatey=coordinatey1

        elif (mu>ss1 and mu<=ss2):
                aux=(mu-ss1)/(ss2-ss1)
                coordinatex=(1.-aux)*coordinatex1+aux*coordinatex2
                coordinatey=(1.-aux)*coordinatey1+aux*coordinatey2

        elif (mu>ss2 and mu<=ss3):
                aux=(mu-ss2)/(ss3-ss2)
                coordinatex=(1.-aux)*coordinatex2+aux*coordinatex3
                coordinatey=(1.-aux)*coordinatey2+aux*coordinatey3

        elif (mu>ss3 and mu<=ss4):
                aux=(mu-ss3)/(ss4-ss3)
                coordinatex=(1.-aux)*coordinatex3+aux*coordinatex4
                coordinatey=(1.-aux)*coordinatey3+aux*coordinatey4

        elif (mu>ss4 and mu<=ss5):
                aux=(mu-ss4)/(ss5-ss4)
                coordinatex=(1.-aux)*coordinatex4+aux*coordinatex5
                coordinatey=(1.-aux)*coordinatey4+aux*coordinatey5

        elif (mu>ss5):
                coordinatex=coordinatex5
                coordinatey=coordinatey5
        else:
                coordinatex=coordinatex5
                coordinatey=coordinatey5
```

```
        return coordinatex, coordinatey

##=============================================================


##main
## load airfoil to define number of points per airfoil
[basex, basey]=read_airfoil("naca5425.dat")



[coordinatex1, coordinatey1]=read_airfoil("naca5425.dat") ## airfoil at mu=0.1
ss1=0.01
[coordinatex2, coordinatey2]=read_airfoil("naca5425.dat") ## airfoil at mu=0.1
ss2=0.03
[coordinatex3, coordinatey3]=read_airfoil("naca5425.dat") ## airfoil at mu=0.1
ss3=0.1
[coordinatex4, coordinatey4]=read_airfoil("naca5425.dat") ## airfoil at mu=0.3
ss4=0.3
[coordinatex5, coordinatey5]=read_airfoil("naca6618.dat") ## airfoil at mu=0.8
ss5=0.8


Diam=98.8 #diameter
R=Diam/2


alpha=2        ## angle of attack
af=alpha*pi/180
windspeed=12.1
ldesign=7.5        ## design tip speed ratio

nz=20.
r=1/nz
mu=arange(0.01,0.9999,r)        ## mu=rreal/R
rreal=R*mu

N=3.                ## number of blades
ai=0.3                ## initial flow idunction parameter


pi=math.pi
```

```
e=math.e

Cl=1.05        ## Cl
file_coord='icemcoord2.txt'
npoints=len(coordinatex1)/2
nlines=len(mu)*2


## initializations
w=write_points(file_coord, 0,0,0,0,npoints,nlines)

a=zeros((len(mu),1))
aa=zeros((len(mu),1))
expo=zeros((len(mu),1))
f=zeros((len(mu),1))
fp=zeros((len(mu),1))
c=zeros((len(mu),1))
coordinates =[]
twistcoordinates =[]

for i in range(1, len(mu)+1):
        a[i-1]=ai
        expo[i-1]=(e**(-((N/2.)*(1.-mu[i-1])/mu[i-1])*sqrt(1+
                        (ldesign*mu[i-1]**2)/(1.- a[i-1]**2))))
        faux=(2./pi)*math.acos(expo[i-1])        ## tip loss factor
        f[i-1]=faux
        ## flow induction factor
        a[i-1]=(1./3)+(1./3.)*f[i-1]-(1./3.)*sqrt(1.-f[i-1]+f[i-1]**2)
        aa[i-1]=a[i-1]*(1.- a[i-1]/f[i-1])/(ldesign**2*mu[i-1]**2)

        ev=1
        while ev>0.0001:
                fp[i-1]=f[i-1]
                expo[i-1]=e**(-((N/2.)*(1.-mu[i-1])/mu[i-1])*sqrt(1+(ldesign*
                                mu[i-1]**2)/(1.- a[i-1]**2)))
                f[i-1]=(2./pi)*math.acos(expo[i-1])
                aa[i-1]=a[i-1]*(1.- a[i-1]/f[i-1])/(ldesign**2*mu[i-1]**2)
                ev=abs((fp[i-1]-f[i-1])/f[i-1])


## calculating chord and airfoil data
```

```
        chord=(R*8/(9*0.8*ldesign))*(2-ldesign*mu[i-1]/(ldesign*0.8))
                                                    *(2*pi)/(Cl*ldesign*N)


        [coordinatex, coordinatey]=recalc_airfoil(coordinatex1, coordinatey1,
coordinatex2, coordinatey2, coordinatex3, coordinatey3, coordinatex4,
coordinatey4, coordinatex5, coordinatey5, ss1, ss2, ss3, ss4, ss5, mu[i-1])

## applying chord to airfoil calculated
        [x,y]=aply_chord(chord,coordinatex,coordinatey)


        z=ones((len(x),1))*rreal[i-1] ## z raio


## twist calculations

        phi=arctan((1-a[i-1])/(ldesign*mu[i-1]*(1.+aa[i-1])))
        tetha=af-phi
        c=math.cos(tetha)
        s=math.sin(tetha)
        Tmat = mat(        [[c,s,0.],
                        [-s,c,0.],
                        [0.,0.,1.]])
        gama=-pi/2
        c=math.cos(gama)
        s=math.sin(gama)
        Mrot = mat(        [[1,0.,0.],
                        [0,c,-s],
                        [0.,s,c]])


        disp=5.

        for j in range(0,len(x)):

                coordinates=mat([float(x[j]),float(y[j]),float(z[j])])
                co= Tmat*coordinates.T ## rotation to apply twist
                coord= Mrot*co ## rotation to the defined referential
                ## write coordinates to file
                w=write_points(file_coord, coord[0], coord[1]+disp, coord[2],w)
```

```
                twistcoordinates.append(coord) ## salva lista com as coordenadas


os.system('python fluidout.py')
os.system('python bladeout.py')

# define a new FSI case
os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
                                        'rm -R $FOAM_RUN/newcasefsi001\n')
os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
'mkdir $FOAM_RUN/newcasefsi001\n'+
'cp -r $FOAM_RUN/newcasefsi/fluid $FOAM_RUN/newcasefsi001\n'+
'cp -r $FOAM_RUN/newcasefsi/solid $FOAM_RUN/newcasefsi001\n')

# copy the created meshes to the case folder
os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
'cp fluentfluid.msh $FOAM_RUN/newcasefsi001/fluid/fluentfluid.msh\n'+
'cp fluentblade.msh $FOAM_RUN/newcasefsi001/solid/fluentblade.msh\n')

# convert to OpenFoam format and delete the trace
os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
'cd $FOAM_RUN/newcasefsi001/fluid/\n'+'fluent3DMeshToFoam fluentfluid.msh\n'+
'createPatch\n'+'rm -R constant/polyMesh\n'+
'cp -r 1e-06/polyMesh constant/\n'+'rm -r 1e-06\n')

os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
'cd $FOAM_RUN/newcasefsi001/solid/\n'+
'fluent3DMeshToFoam fluentblade.msh\n')
os.system('. /usr/lib/OpenFOAM-1.6-ext/etc/bashrc\n'+
'cd $FOAM_RUN/newcasefsi001/fluid/\n'+'simpleFsiFoam\n')
```

A.2 Definition of the Geometry of the Structural Model

Example of one of the scripts to export commands to ICEMCFD.

```
import os
import string
from math import *
from numpy import *
import scipy as Sci
import scipy.linalg
```

```
def write_to_file(filename, string, w):

        if w == 0:
                    fout=open( filename, 'w' )
                fout.write( string)
         else:
                fout=open( filename, 'a' )
                fout.write( string)

        return 1

def write_spline(filename, p_i, p_f,l,inc):
        if (p_i<p_f):
                s='ic_curve point CRVS crv.%d  { '%(l)
                for i in arange( p_i, p_f , inc):
                        d='pnt%d '%(i)
                        s=s+d

                s=s+'pnt%d }\n'%(p_f)

                w=1
                write_to_file(filename,s,w)

        return l


def line_(filename, p_i, p_f,l):
        s='ic_curve point CRVS crv.%d  { pnt%d pnt%d }\n'%(l,p_i,p_f)
        w=1
        write_to_file(filename,s,w)
        return l

def line2(filename, p_i, p_f,l):
        s='ic_curve point CRVS crv.%d  { pnt%d pnt%d }\n'%(l,p_i,p_f)
        w=1
        write_to_file(filename,s,w)
        return l



def write_surface(filename, l_i,sf, part,l2=None,l3=None,l4=None):
```

```
        if l2==None and l3==None and l4==None:
                s="ic_surface 2-4crvs "+ part +
" srf.%d {0.001 {crv.%d crv.%d crv.%d crv.%d}}\n"%(sf,l_i,l_i+11,l_i+22,l_i+23)
        elif l4==None:
                s="ic_surface 2-4crvs "+ part +
" srf.%d {0.001 {crv.%d crv.%d crv.%d}}\n"%(sf,l_i,l2,l3)
        else:
                s="ic_surface 2-4crvs "+ part +
" srf.%d {0.001 {crv.%d crv.%d crv.%d crv.%d}}\n"%(sf,l_i,l2,l3,l4)
        w=1
        write_to_file(filename,s,w)

        return sf
####============================================================================#####

#main
rplfile='windbladesolid.rpl'
w=0
l=0

s='# ficheiro de replay para o icem que faz a malha da pa\n'
w=write_to_file(rplfile,s,w)

s=('ic_geo_cre_geom_input /home/brunotojo/py/icemcoord2.txt'+
'0.001 input PNTS pnt CRVS {} SURFS {}\n')
w=write_to_file(rplfile,s,w)

f=0
l=write_spline(rplfile,0+f,5+f,l+1,1)
l=write_spline(rplfile,5+f,10+f,l+1,1)
l=write_spline(rplfile,10+f,15+f,l+1,1)
l=write_spline(rplfile,15+f,20+f,l+1,1)
l=write_spline(rplfile,20+f,25+f,l+1,1)
l=write_spline(rplfile,25+f,30+f,l+1,1)
l=write_spline(rplfile,30+f,35+f,l+1,1)
l=write_spline(rplfile,35+f,40+f,l+1,1)
l=write_spline(rplfile,40+f,45+f,l+1,1)
l=write_spline(rplfile,45+f,50+f,l+1,1)
l=line_(rplfile,0+f,50+f,l+1)
```

```
f=51*19  ##number of point in the airfoil * number of airfoils -1

l=write_spline(rplfile,0+f,5+f,l+1,1)
l=write_spline(rplfile,5+f,10+f,l+1,1)
l=write_spline(rplfile,10+f,15+f,l+1,1)
l=write_spline(rplfile,15+f,20+f,l+1,1)
l=write_spline(rplfile,20+f,25+f,l+1,1)
l=write_spline(rplfile,25+f,30+f,l+1,1)
l=write_spline(rplfile,30+f,35+f,l+1,1)
l=write_spline(rplfile,35+f,40+f,l+1,1)
l=write_spline(rplfile,40+f,45+f,l+1,1)
l=write_spline(rplfile,45+f,50+f,l+1,1)
l=line_(rplfile,0+f,50+f,l+1)

l=write_spline(rplfile,0,0+f,l+1,51)
l=write_spline(rplfile,5,5+f,l+1,51)
l=write_spline(rplfile,10,10+f,l+1,51)
l=write_spline(rplfile,15,15+f,l+1,51)
l=write_spline(rplfile,20,20+f,l+1,51)
l=write_spline(rplfile,25,25+f,l+1,51)
l=write_spline(rplfile,30,30+f,l+1,51)
l=write_spline(rplfile,35,35+f,l+1,51)
l=write_spline(rplfile,40,40+f,l+1,51)
l=write_spline(rplfile,45,45+f,l+1,51)
l=write_spline(rplfile,50,50+f,l+1,51)

l=line_(rplfile,5,45,l+1)
l=line_(rplfile,10,40,l+1)
l=line_(rplfile,15,35,l+1)
l=line_(rplfile,20,30,l+1)

l=line_(rplfile,5+f,45+f,l+1)
l=line_(rplfile,10+f,40+f,l+1)
l=line_(rplfile,15+f,35+f,l+1)
l=line_(rplfile,20+f,30+f,l+1)

sf=0
for i in range(2,5):
        sf= write_surface(rplfile,i,sf+1,'SIDEWALLS')
```

```
for i in range(7,10):
        sf= write_surface(rplfile,i,sf+1,'SIDEWALLS')


sf= write_surface(rplfile,13,sf+1,'SIDEWALLS',20,38,39)
sf= write_surface(rplfile,14,sf+1,'SIDEWALLS',19,39,40)
sf= write_surface(rplfile,15,sf+1,'SIDEWALLS',18,40,41)

sf= write_surface(rplfile,37,sf+1,'SIDEWALLS',29,41,27)
sf= write_surface(rplfile,24,sf+1,'SIDEWALLS',34,32,38)
sf= write_surface(rplfile,37,sf+1,'INFF',7,4,36)
sf= write_surface(rplfile,36,sf+1,'INFF',8,35,3)
sf= write_surface(rplfile,35,sf+1,'INFF',9,34,2)


s='ic_geo_new_family BLADE\nic_geo_create_volume {1.5 6.5 -1} {} BLADE\n'
w=write_to_file(rplfile,s,w)
##end of the blade script


s='# fim do ficheiro de script\n\n'
w=write_to_file(rplfile,s,w)

os.system('icemcfd -script '+rplfile+'\n')
```

Appendix B

Script to calculate Cylindric Velocity

File ”myconvertToCylindrical.C”

```
#include "fvCFD.H"
#include "cylindricalCS.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    timeSelector::addOptions();

    #include "setRootCase.H"
    #include "createTime.H"

    instantList timeDirs = timeSelector::select0(runTime, args);

    #include "createMesh.H"

    forAll(timeDirs, timeI)
    {
        runTime.setTime(timeDirs[timeI], timeI);
        Info<< "Time = " << runTime.timeName() << endl;
        mesh.readUpdate();
        // defining cylindrical coordinate system

        Info<< " creating cylindrical system (r, teta, z)
                    from cartesian system with r = y and z = x "<< endl;

        cylindricalCS cyl(
                "cylindricalCS",
                point(0, 0, 0),
```

```
            vector(1, 0, 0),
            vector(0, 1, 0),
            false); //keyword false: Use radians since cos and sin work with radians

        //Read the vectors pointing from the origin of the mesh to the cell centers
        volVectorField cc
        (
            IOobject
            (
                "cc",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            mesh.C()
        );

        //The vector field cc will now be transformed into cylindrical coordinates
        volVectorField ccCyl
        (
                IOobject
                (
                    "ccCyl",
                    runTime.timeName(),
                    mesh,
                    IOobject::NO_READ,
                    IOobject::NO_WRITE
                ),
                mesh,
                vector (0,0,0)
        );
        ccCyl.internalField() = cyl.localVector(cc.internalField());
        forAll (ccCyl.boundaryField(), patchI)
        {
                ccCyl.boundaryField()[patchI] = cyl.localVector(cc.boundaryField()[patchI]);
        }
//      cc.write();
//      ccCyl.write();

        // Define theta
        volScalarField theta
```

```
(
    IOobject
    (
        "theta",
        runTime.timeName(),
        mesh,
```

Appendix C

Script to calculate Power Output

File "Plotpower"

```
#!/bin/bash
truncationFactor=10
name='forces.dat'
# concatenate the diferent forces.dat files,case that simulation was stoped/restarted
cat 0/$name 1/$name 2/$name 4/$name 5/$name 6/$name>$name
# open the file and cut the number of lines defines in truncationFactor
y=$(wc $name | cut -d' ' -f3)
start=$[y/truncationFactor]
echo "Lines in force file: $y"
echo "Line to start plot: $start"

# divide file by colums
sed 's/[()]//g' < $name > processed.dat
sed -i "1,${start}d" processed.dat
# call gnunplot to plot the charts.
gnuplot forcesProcess -
```

File "powerProcess"

```
# wind turbine data
omega=15.11*2*pi/60
Diam=98.8+5
U=12.1
R=Diam/2
# calculate the theoritical available power
P=0.5*1.225*pi*R*R*U*U*U

time=1
xForcePressure=2
xForceViscous=xForcePressure+3
```

```
yForcePressure=3
yForceViscous=yForcePressure+3
zForcePressure=4
zForceViscous=zForcePressure+3
xMomentPressure=8
xMomentViscous=xMomentPressure+3
yMomentPressure=9
yMomentViscous=yMomentPressure+3
zMomentPressure=10
zMomentViscous=zMomentPressure+3


# calculate and plot Power
set term x11 0
set title "Generated␣Power␣from␣FSI␣simulation"
set xlabel "Time"
set ylabel "Power"

plot     "processed.dat" using 1:(($8+$11)*omega) title 'Power␣(Watts)' \

# calculate and plot Thust
set term x11 1
set title "Thrust/Drag␣from␣FSI␣simulation"
set xlabel "Time"
set ylabel "Force"

plot    "processed.dat" using 1:(($2+$5)) title 'Thrust/Drag␣(Newtons)'\


# calculate and plot power coeficient
set term x11 2
set title "CP␣from␣FSI␣simulation"
set xlabel "Time"
set ylabel "Cp"

plot     "processed.dat" using 1:(($8+$11)*omega/P) title 'Power␣Coeficient' \
```