Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

On the design of a toroidal wind turbine

Felipe Andrade

PROJETO DE GRADUAÇÃO

ENGENHARIA MECÂNICA

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Sobre o projeto de turbinas eólicas toroidais: geração de geometria

Felipe Andrade

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro Mecânico

Orientador: Prof. Dr. Adriano Possebon
Coorientador: Prof. Dr. Antonio Brasil Jr.

Brasília
2023

Andrade, Felipe.
A769o On the design of a toroidal wind turbine / Felipe Andrade; orientador Adriano Possebon; coorientador Antonio Brasil Jr.. -- Brasília, 2023.
94 p.

Projeto de Graduação (Engenharia Mecânica) -- Universidade de Brasília, 2023.

1. Turbina eólica. 2. Blade Element Momentum. 3. Turbina toroidal. 4. Desenho de Rotores. I. Possebon, Adriano, orient. II. Brasil Jr., Antonio, coorient. III. Título

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

On the design of a toroidal wind turbine

Felipe Andrade

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro Mecânico

Trabalho aprovado. Brasília, 20 de dezembro de 2023:

Prof. Dr. Adriano Possebon Rosa,
UnB/FT/ENM
Orientador

Prof. Dr. Rafael Castilho Faria Mendes,
UnB/FGA
Examinador interno

Prof. Dr. Bráulio Gutierrez Pimenta,
UnB/FT/ENM
Examinador externo

Ramiro de Matos Bertolina
Examinador interno

Brasília
2023

Dedico este trabalho a minha família,
e a todos que com sabedoria e amor
fazem o possível e o impossível
para tornar o mundo um lugar melhor.

Acknowledgements

Gostaria primeiramente de agradecer à minha família que me apoiou na jornada da obtenção do diploma de Engenheiro Mecânico, aos meus amigos e colegas que me ajudaram e tornaram a experiência mais leve. Gostaria de agredecer aos meus orientadores, Prof. Adriano Possebon e Prof.Antônio Brasil Jr., que me guiaram e me ensinaram muito durante não só o desenvolvimento deste trabalho, como também durante a graduação, me ofereceram oportunidades únicas de realizar pesquisas, a prestar monitorias e em suma a crescer pessoalmente e profissionalmente. Tendo trabalhado com ambos, reconheço a grande competência, e conhecimento dos dois em suas respectivas áreas de trabalho, reconheço-os como bons professores, bons orientadores e mais importante boas pessoas. Nem sempre nossa relação foi fácil, mas sempre foi produtiva e enriquecedora, e por isso faço questão de expressar minha gradidão para além das formalidades.

Gostaria de agradecer também ao Prof. Taygoara Felamingo, que me ensinou e me ajudou a compreender melhor a Dinâmica dos Fluidos, a também o Prof. Rafael Mendes, quem sempre me ajudou durante todo meu tempo no Laboratório de Energia e Ambiente, a Matheus Nunes, quem me ajudou a compreender melhor o funcionamento do OpenFOAM, e quem me ajudou a realizar as simulações da geometria gerada neste trabalho, bem como Ramiro Bertonila, quem me ajudou bem na minha entrada no laboratório a entender como gerar malhas.

Além disso, gostaria de agradecer ao Prof. Bráulio Pimenta, quem me ajudou a entender melhor a aerodinâmica e se dispôs por vezes a auxiliar no projeto. Ao Prof. Roberto Miserda, quem me ajudou a entender melhor o método dos volumes finitos. Ao Prof. Rafael Gabler e ao Prof. José Luiz da Fontoura, que me ajudaram a compreender mesmo que superficialmente a teoria de turbulência. E ao Prof. Antônio Manuel Dias, quem me deu lições valiosas sobre a execução de projetos.

Por fim, gostaria de agradecer a todos aos professores que tive durante a graduação, que me ensinaram e me ajudaram a crescer, e a todos os funcionários da FT, que sempre me ajudaram e me apoiaram durante todo o curso.

"It was possible, no doubt, to imagine a society in which wealth, in the sense of personal possessions and luxuries, should be evenly distributed, while power remained in the hands of a small privileged caste. But in practice such a society could not long remain stable. For if leisure and security were enjoyed by all alike, the great mass of human beings who are normally stupefied by poverty would become literate and would learn to think for themselves; and when once they had done this, they would sooner or later realize that the privileged minority had no function, and they would sweep it away. In the long run, a hierarchical society was only possible on a basis of poverty and ignorance"

"Talking to her, he realized how easy it was to present an appearance of orthodoxy while having no grasp whatever of what orthodoxy meant. In a way, the world-view of the Party imposed itself most successfully on people incapable of understanding it. They could be made to accept the most flagrant violations of reality, because they never fully grasped the enormity of what was demanded of them, and were not sufficiently interested in public events to notice what was happening. By lack of understanding they remained sane. They simply swallowed everything, and what they swallowed did them no harm, because it left no residue behind, just as a grain of corn will pass undigested through the body of a bird"

(George Orwell, "1984", 1949)

Abstract

This work presents a numerical and algorithmical study of a toroidal turbine rotor design generation. The toroidal design has been shown as a propeller to allow for operation in a wider range of angular velocities without the disadvantages of cavitation, reduced trailing vortices, and noise production. Hence, it presents higher efficiency and power coeficients compared to conventional designs. For that reason, it can not only operate in a larger variety of hydrological and meteorological conditions but also diminish disturbances in the natural life surrounding it.

The main objectives of this research are to generate a toroidal rotor for axial flow turbines based on the classic Blade Element Momentum Theory and to preliminarily evaluate the performance of the newly designed rotor through numerical simulations. The lift and drag coefficients table were calculated using XFOIL. The algorithms of the Blade Element Momentum Method and the necessary geometric manipulation of the geometry were implemented, and preliminary results were obtained.

The findings of this study have relevance in furthering the knowledge on toroidal rotors and pratical applications in the design and optimization of toroidal axial flow turbines employed in both hydrokinetic and wind power generations. Therefore, it contributes to the development of more environmentally friendly and efficient energy sources.

Keywords: Wind turbine. Blade Element Momentum. Toroidal Turbine. Rotor Design.

Resumo

Este trabalho apresenta um estudo numérico e algorítmico do projeto de um rotor toroidal para turbinas axiais horizontais. A geometria toroidal de propulsores tem permitido a operação em uma faixa de velocidades angulares mais ampla reduzindo problemas de cavitação, vórtices de ponta de pá e produção de ruído. Portanto, ela apresenta maior eficiência e coeficientes de potência em comparação com geometrias convencionais. Por tal motivo, ela não apenas pode operar em uma variedade maior de condições hidrológicas e meteorológicas, mas também diminuir perturbações na natureza ao seu redor.

O principal objetivo desta pesquisa é gerar um rotor toroidal para turbinas axiais baseado na teoria clássica de Blade Element Momentum e avaliar seu desempenho através de simulações numéricas preliminares. A tabela de coeficientes de sustentação e arrasto foram obtidas utilizando o XFOIL.Os algoritmos do método Blade Element Momentum e a manipulação geométrica necessária da geometria foram implementados.

As descobertas neste trabalho são relavantes no que se concerne a expanção da fronteira de conhecimento sobre rotores toroidais e na aplicação prática do projeto e otimização de turbinas axiais toroidais empregadas tanto na geração hidrocinética quanto eólica. Portanto, contribui para o desenvolvimento de fontes de energia mais eficientes e sustentáveis.

Palavras-chave: Turbina eólica. Blade Element Momentum. Turbina toroidal. Desenho de Rotores.

List of Figures

List of Tables

Contents

1 Introduction

1.1 The Climate Crisis

At the beginning of the XXI century, the greatest problem to be faced by human civilization is climate change.

Although there are other big problems that humanity needs to address such as poverty, health crises, violence, hunger, population growth, and innumerable others, the single problem with has the most potential to impact the human experience is undoubtedly the climate crisis. The climate crisis is highly correlated with the increase in global CO_2 emissions since the Industrial Revolution, creating the projected scenario of a 2ºC increase in the global average temperature.

According to Breakthrough Energy (2023), electricity production contributes as much as 27% of global emissions, reinforcing the urgency of changing our energetic matrix to something more sustainable. It can be therefore said, that the sustainable energy sources technologically currently viable are mainly: solar, nuclear, wind, and hydrokinetic energy. This study focuses on the study of horizontal axial wind turbines, which are the most common wind turbine designs that can be seen around.

1.2 Wind Energy

> A remembrance of the historical roots of wind turbine technology is in the context of the operation of modern wind turbines is more than just a weekend reading. The technical solutions and the economic conditions, which have brought success and failure, give us hints about advancements made today and in the future.

After the Great War came to an end, engineers started using the advancements in aeronautics to build wind turbines and began using aerodynamic profiles to build their blades (GIPE; MÖLLERSTRÖM, 2022). This new design allowed for a lift-based turbine design making them more efficient and less prone to over-rotation in extreme weather due to stall.

Both the influence of aeronautics and the necessity of new energy sources were also remarked by Betz nearly over a century ago.

> The attention paid to wind turbines has considerably increased in the post-war era. Fundamentally there must be two reasons for the increased interest in the topic: First the efforts in finding new energy sources in light of the coal scarcity. Then there is probably the feeling that the experience brought by

> flight techniques has the possibility of offering fundamentally the perfection of wind turbines.

In the cited article, Betz has proposed the so-called Betz Limit, which is fundamentally the efficiency limit of wind turbines. For this purpose, Betz used the Momentum Theory, first proposed by Froude (1920) while studying ship propellers at his time, a theory which was called the Screw Propeller Theory, developed the first steps to what would later become the Blade Element Momentum Theory.

The Blade Element Momentum Theory (BEMT), refined by Glauert (1935) uses simplifications to the flow around the rotor including:

- The independence of blade elements, in other words, the phenomena affecting one element will not affect neighboring elements;
- Constant forces in each radial position. Along with the Momentum Theory, these simplifications allow for a relatively easy calculation of important properties for instance the optimal angle of attack and chord length in any radial position, the expected thrust, torque, and consequently the predicted power coefficient and efficiency;
- The flow is also considered incompressible and inviscid.

These concepts will be further discussed in this work.

It is important to highlight that advancements made in the design of propellers were always adapted and used in the development of better wind turbines, due to their similarities. Little has changed since then in the rotor design methods, the overall shape of the blades has stayed pretty much the same, and the current design is close to the the theoretical peak of efficiency.

1.3 Wake Aerodynamics

The wake of a wind turbine is highly complex, in the sense that there are different forces acting upon the behavior of the flow. This is due to the complex nature of the flow in the rotor area itself, where pressure, centrifugal and Coriolis forces are into play, not to mention the production of vorticity and turbulence due to the viscous interactions with the blade and eventual separation of the boundary layer which leads to stalling Vermeer (2003).

The wake can be divided into two separate regions The Near Wake and the Far Wake (VERMEER; SØRENSEN; CRESPO, 2003):

- **Near Wake:** can be defined as the region right behind the rotor up to one rotor diameter downstream, there is a high influence of the rotor geometry in the near wake, because of aspects such as the number of blades, solidity, and tip geometry are apparent.

- **Far Wake:** on the other hand is the region from one rotor diameter up to approximately to ten rotor diameters, the exact distance from the rotor's outlet area to the end of the far wake is defined by the rate of pressure restitution of the flow. In this region effects such as wing tip and trailing vortices are broken down by the effect of vortex stretching.

In Figure (1.1) it can be observed the different regions of the wake of a wind turbine including the expansion of the streamlines, the presence of tip vortices, the turbulent mixing region, where vortices are broken down, and the far wake where there is the recovery to the state of upwind from the rotor.

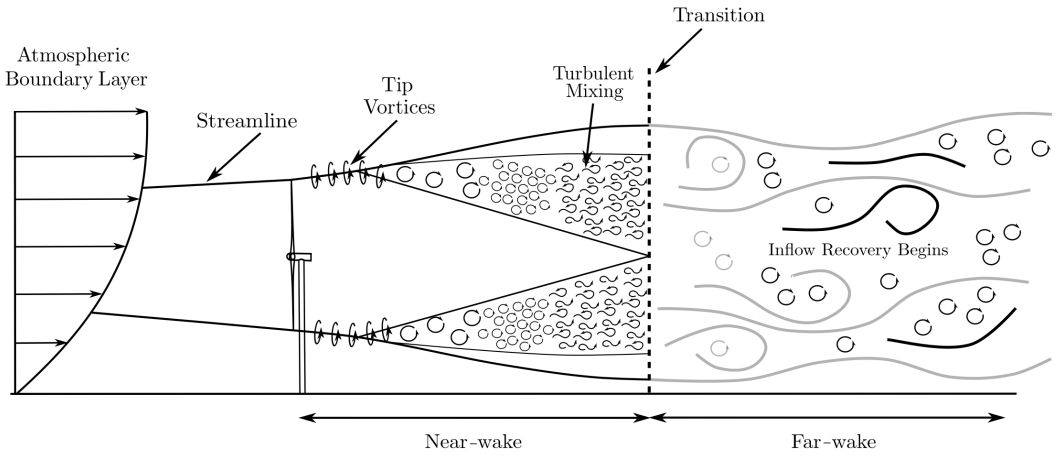


Figure 1.1 – Example of the regions of the wake of a wind turbine (RODRIGUEZ; JAWORSKI; MICHOPOULOS, 2021)

Figure (1.2) portrays the vortex structures in the near wake of the rotor of a wind turbine. The root, tip, and trailing vortices are visible in the figure.

1.3.1 Wing Tip Vortices

Wing tip vortices are produced as a consequence of the pressure differential of the upper part and lower part of an aerofoil. Owing to the angle of attack of the airfoil with the mean flow, a high-pressure zone is generated on the pressure side (lower surface), while a low-pressure zone is generated on the suction side (upper surface). On the tip, those different pressure zones aren't bounded by neighboring solid elements, which means that the fluid from the high-pressure zone is free to move itself to the low-pressure zone, this circular motion generates a vortex which due to the direction of the flow will be advected to the free flow downstream.

Wing tip vortices are advected conservatively through the fluid because the dissipation effects are of lesser magnitude in comparison to the forces generated by the presence of

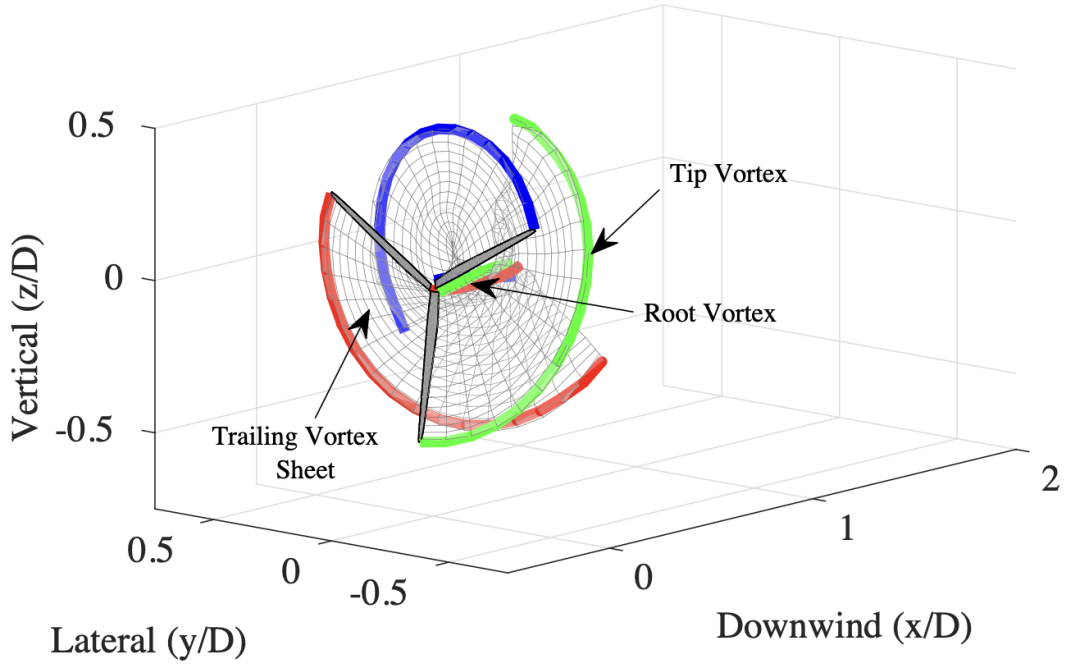


Figure 1.2 – Vortex Structures in the Near Wake illustrated by (RODRIGUEZ; JAWORSKI; MICHOPOULOS, 2021)

the pressure gradient, therefore it may be studied in the realm of validity of the potential theory, where the pressure may be considered a simple potential solenoidal field.

As in any fluid in rotating motion, that is vortices, the pressure in the middle of it is lower than on the outside, which generates and maintains the structure of the vortices. This generated pressure gradient is the main source of noise production because the lower pressure zone will inevitably generate pressure waves and carry sound through the fluid domain.

Additionally, when turbines and propellers work in water, wing tip vortices and their center low-pressure zone cause an effect called cavitation. In addition to the loss of energy from not only rotating the fluid but also changing the phase of the fluid, the cavitation phenomenon also releases a great amount of energy which poses a problem of corrosion in the rotor blades naturally decreasing its work span, hence representing a great challenge in the design of hydrokinetic turbines.

Many attempts have been made to reduce the formation of these vortices, such as the use of end plates in hydrokinetic turbines (HAU, 2017), however, the end-plates also generate increased inertia, which leads to higher activation speeds, and drag, which also dissipates energy from the fluid.

1.3.2 Trailing Vortices

Trailing vortices are naturally occurring because of the difference in pressure between the suction and pressure of the airfoil, and the velocity deficit in the boundary layer which generates vorticity and instabilities, causing fluctuations in the pressure field, hence noise production.

In Figure (1.3) the streamlines of the flow over an airfoil is illustrated, it can be seen that the streamlines do not close themselves in the trailing edge, and it can be seen that the boundary layer affects the formation of the trailing vortices.

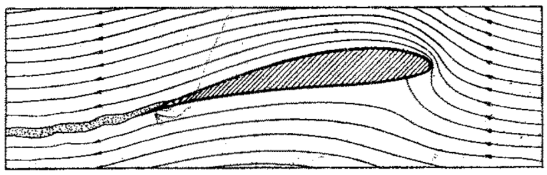


Fig. 7. Wirkliche Strömung um ein Flügelprofil. Die Strömung schließt sich an der Hinterkante nicht mehr vollständig zusammen. Es bleibt ein „Totwasser". Hierdurch wird die Zirkulation und damit der Auftrieb kleiner als bei der theoretischen Potentialströmung (Fig. 3).

Figure 1.3 – "Real flow over an airfoil. The stream lines do not close themselves in the trailing edge. A 'dead water' (*Totwasser*) remains. Hereby the circulation and along with it the lift is smaller then by the Potential Theory" (BETZ, 1918)

1.4 Toroidal Geometry

In the beginning of 2023, a patent from MIT (SEBASTIAN; STREM, 2019), has been publicly published regarding the design of a rotor with toroidal-shaped blades. Those blades were designed to be used as propellers in drones, and have been shown to reduce the audible noise generated by the rotation of the rotor, as observed in Figure (1.4).

Additionally, the company Sharrow Marine, which produces boat propellers has started selling rotors with an adapted toroidal geometry, which has shown to reduce not only noise production but also cavitation (the formation of bubbles of air due to an extreme decrease in pressure causing the phase transition of water). Sharrow Marine (2022) has also reported an increase in the efficiency of boats with the use of this rotor as it may be seen in Figure (1.5) published in their performance report.

As of the date of writing of this work, no academic articles or research results have been published about the toroidal geometry, and from a scientific standpoint the study of this geometry and the flow impacted by it is justified.

The results presented both by the MIT Lincoln Laboratory and Sharrow Marine look promising in addressing the main problems with axial flux machines namely noise

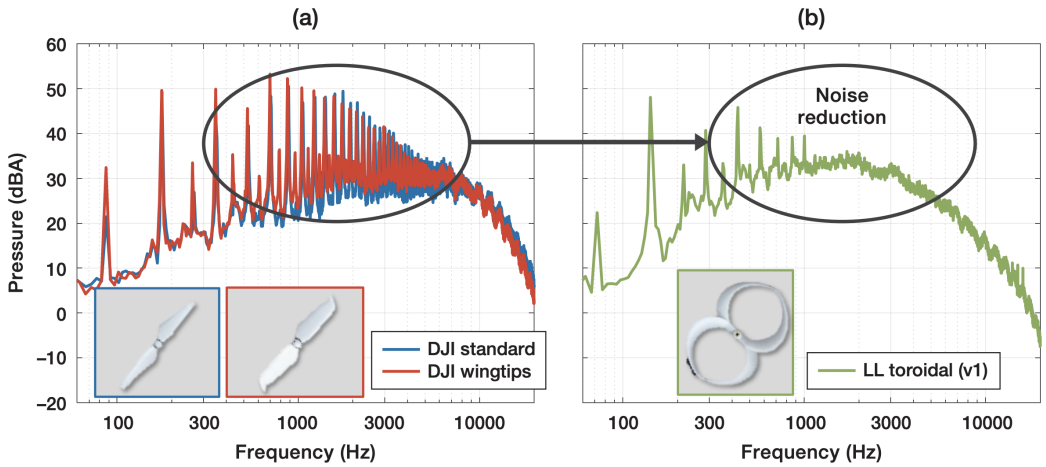


Figure 1.4 – Comparison between the noise produced by a quadrotor with traditional blades (a) and with toroidal blades (b) (LINCOLN LABORATORY - MIT, 2022)

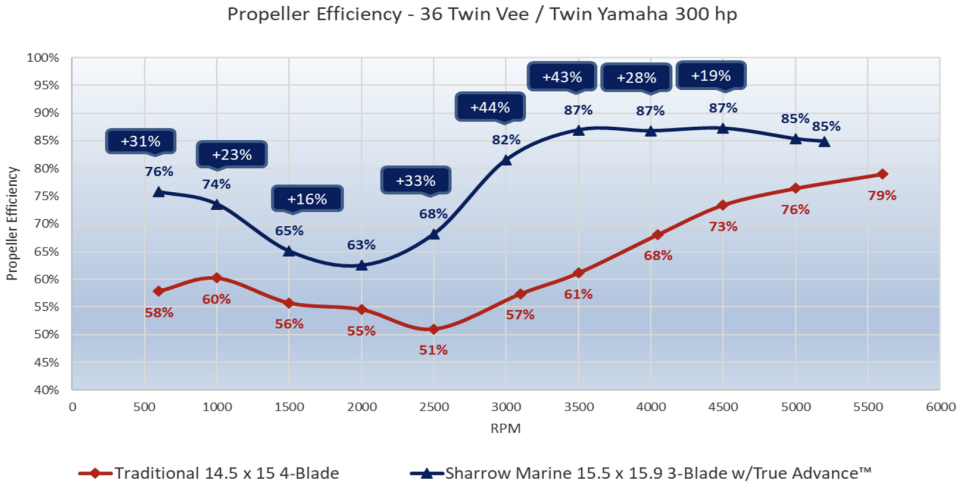


Figure 1.5 – Reported Efficiency comparison between standard propellers and Sharrow Marine's toroidal-shaped propellers (SHARROW MARINE, 2022)

production and cavitation, and the production of wing tip vortices. It is only reasonable to hypothesize whether the advantages of the new geometries extend to be used in flux machines to produce electricity. As historically it can be argued that the developments in propeller technologies and turbine rotor technologies walk side by side.

Furthermore, as the wake of this rotor design has never been thoroughly studied, it is of scientific interest to understand how the absence of wing-tip vortices affect the behavior of the wake from the turbine.

1.5 Objectives

The main objectives of this work may be summarized as follows:

1. Generate an algorithm capable of generating an STL file of a toroidal wind turbine rotor, based on the Lincoln Laboratory patent (SEBASTIAN; STREM, 2019) and on BEM;
2. Analyse the pressure distribution on the surface of the rotor;
3. Study the power coefficient of the rotor by varying the tip speed rate (TSR);

2 Theory

2.1 Blade Element Momentum Theory BEMT

The **Momentum Theory** or the **Actuator Disk Theory** is one of the most important theories when it comes to the study and project of wind turbines and other axial flow turbines. It is based on the approximation of the rotor as a permeable disk and the flow as incompressible, steady, inviscid, and axial; Also, mass and momentum are conserved in the flow.

These considerations are necessary for the use of Bernoulli's principles. The momentum theory is used to obtain the power coefficient and the thrust coefficient, the axial and tangential induction factors. These results may be used along with turbomachinery theory to obtain the ideal pitch angle, and the ideal chord length of blade elements, and therefore allowing for the project of an ideal wind turbine rotor.

According to Hansen (2015), thrust is the force resulting from the pressure drop in the rotor, this force reduces the axial velocity of the flow from the upstream velocity u_0 to the downstream velocity u_2. It can be written as

$$T = \Delta p A, \tag{2.1}$$

where Δp is the pressure drop in the rotor, and A is the area of the rotor. The pressure drop may be found via Bernoulli's equation. The thrust coefficient is given by

$$C_T = \frac{T}{\frac{1}{2}\rho u_0^2 A}. \tag{2.2}$$

Wind turbines convert kinetic energy from the wind into mechanical energy via the rotation of the rotor (shaft power). The power coefficient is defined then as the ratio between the available power in the stream and the power extracted by the rotor. The power coefficient may be written as

$$C_P = \frac{P}{P_{\text{avail}}}, \tag{2.3}$$

where P is the power extracted by the rotor, with the available power being

$$P_{\text{avail}} = \frac{1}{2}\rho A u_0^3, \tag{2.4}$$

where A is a cross-section of the flow upstream with the same area as the rotor. Thus,

$$C_P = \frac{P}{\frac{1}{2}\rho A u_0^3}. \tag{2.5}$$

The Bernoulli's principle states that an increase in velocity of the fluid is directly correlated to a decrease in pressure or potential energy. This principle may be mathematically expressed as

$$\nabla_i\left(\frac{\rho u_i^2(x_i)}{2} - p(x_i) + \Phi_i(x_i)\right) = 0 \tag{2.6}$$

where x_i are the points of the coordinate in analysis, and Φ_i is the potential energy term, in general, this term is used to express the gravitational potential energy by unit volume $\Phi_i(x_i) = \rho g x_i$. Other requirements of the Bernoulli equation are:

- The conservation of streamlines (as cited);
- Isentropic Flows, when the flow displays neither irreversibilities such as turbulence nor dissipation such as viscosity or compressibility.

Because the streamlines are not conserved from the front to the back of the rotor, Bernoulli's equation cannot be directly applied, thus it is necessary to indirectly apply it, resulting in the equation system

$$\begin{cases} \frac{\rho u_0^2}{2} + p_0 = \frac{\rho u_1^2}{2} + p_1 \\ \frac{\rho u_1^2}{2} + p_1 - \Delta p = \frac{\rho u_2^2}{2} - p_0, \end{cases} \tag{2.7}$$

solving it for Δp results in

$$\Delta p = \frac{\rho}{2}(u_0^2 - u_2^2). \tag{2.8}$$

According to Betz (1922), the axial induction factor is defined as

$$a = \frac{u_2}{u_0}, \tag{2.9}$$

or for the velocity at the disk (HANSEN, 2015),

$$u_1 = (1 - a)u_0, \tag{2.10}$$

and relating the velocity downstream to the velocity upstream by

$$u_2 = (1 - 2a)u_0. \tag{2.11}$$

The axial induction factor is a measure of the decrease in the axial velocity of the flow due to the presence of the rotor. This happens due to the sole presence of the rotor, on account of the pressure increase near the rotor. This can be thought of as the fluid particles pressing themselves together because of the wall causing an increase in pressure, and concomitantly a decrease in velocity.[1]The tangential induction factor, on the other hand, is caused by the

[1] This is not a valid representation of the flow, because the flow is deemed incompressible, therefore the particles do not press themselves together, rather the pressure from the particles in the rotor wall pushes the particles in the incoming flow away from the wall and for that to happen they need to loose axial velocity.

movement of the blade, which pushes the fluid particles in the tangential direction, due to the increased pressure in the leading edge, therefore the tangential component of the fluid velocity immediately in front of the leading edge will be $(1 + a')\omega r$, for ω_r being the tangential velocity of the blade section at the radial position r.

Using mass and momentum conservation principle over an arbitrary control volume, along with the axial induction factor, it is possible to obtain the thrust and power coefficient as a function of a. Which may be written as

$$C_T = 4a(1 - a), \tag{2.12}$$

for the thrust coefficient, and

$$C_P = 4a(1 - a)^2, \tag{2.13}$$

for the power coefficient.

The Betz Limit is the point of maximal power coefficient for an axial flow machine, which can be found by taking the derivative of the power coefficient in relation to the induction factor and equaling it to zero, resulting in

$$\frac{dC_P}{da} = 4(1 - 3a)(1 - a) = 0, \tag{2.14}$$

which results in either $a = 1$, which is not a physical solution, because if $a = 1$ then $u_2 = 0$, and the flow would be stopped thus not extracting any power from the flow, or $a = 1/3$, which is the Betz Limit.

Some important remarks must be made, first of all, the more C_T increases, the more the streamlines are deflected by the presence of the rotor, as a means of conserving mass. This deflection creates a larger area behind the rotor. Secondly, increasing C_T for $a > 1/4$ leads to a large pressure drop, which induces the outer flow to transfer momentum to the wake-generating eddies, a state that is called a turbulent wake state. Therefore the momentum theory is not valid for $a > 1/4$.

In non-ideal rotors the tangential is not zero, due to the rotation of the rotor, which generates an opposite rotation in the flow, thus generating a tangential velocity. This can be better illustrated by the velocity triangle in a profile section of the rotor, as shown in Figure (2.6).

Figure (2.6) is a typical velocity triangle for an axial flow machine, wherein the leading edge, V_{rot} is the rotational velocity at the profile section, that is $V_{rot} = \omega r$, u is the inflow velocity at the rotor intake, and $V_{rel,1}$ is the fluid's velocity as seen by a moving observer, stationary in relation to the blade with magnitude $||V_{rel,1}|| = \sqrt{V_{rot}^2 + u^2}$. As for the trailing edge, it is known due to the Kutta Condition, that $V_{rel,2}$ approximately follows the trailing edge for small angles of attack and flows with little to no boundary layer separation, due to viscosity (ANDERSON, 2011). On the other hand, $\mathbf{C} = (C_r, C_\theta, C_a)$ is the velocity

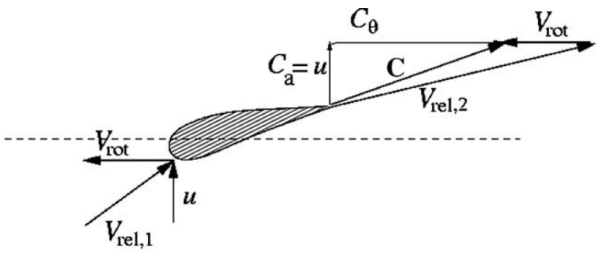


Figure 2.6 – Velocity triangle in a profile section of the rotor. (HANSEN, 2015)

observed by an inertial observer, where C_r is the radial velocity, C_θ is the tangential velocity, and C_a is the axial velocity.

It is desirable to have a large rotational speed for C_θ to be minimal (HANSEN, 2015), as illustrated in Figure (2.6).

Analogously to eq. (2.10), the tangential induction factor a' is given by

$$C_\theta = 2a'\omega r, \tag{2.15}$$

and the infinitesimal power by

$$dP = \frac{1}{2}\rho C_\theta u_1 dr, \tag{2.16}$$

substituting eq. (2.15) in eq. (2.16), using the convenient adimensionalisation namely the tip-speed ratio

$$\lambda(r) = \frac{\omega r}{u_0}, \tag{2.17}$$

with $\lambda(R) = \lambda$ and integrating it over the maximum TSR results in

$$P = \frac{8}{\lambda^2}\int_0^\lambda (1-a)a'dr. \tag{2.18}$$

With eq. (2.18) it is possible to see that to maximize performance, $(1-a)a'$must be maximized. Therefore,

$$[(1-a)a']_{.a} = (1-a)a'_{.a} - a' = 0. \tag{2.19}$$

However, having a look at the velocity triangle with the induction factors in Figure (2.7), it is possible to see that the effective angle of attack ϕ is given by

$$\tan(\phi) = \frac{U_\infty(1-a)}{\Omega r(1+a')} = \frac{(1-a)}{\lambda_r(1+a')}, \tag{2.20}$$

where $\lambda(r) = \lambda_r$, $U_\infty = u_0$, $\Omega = \omega$ and $W = V_{\text{rel},1}$ is the relative velocity at the rotor intake. However, from Figure (2.6) it is possible to see that,

$$\tan(\phi) = \frac{a'\omega r}{au_0} = \frac{a'}{a}\lambda_r,$$

is also true by Glauert's approach of triangle similarity according to (HANSEN, 2015) and (BRASIL JUNIOR et al., 2019), allowing for the conclusion that

$$a'(1+a')\lambda_r^2 = a(1-a), \tag{2.21}$$

with the root of the equation being,

$$a' = \frac{-1+\sqrt{1+4a(1-a)\lambda_r^{-2}}}{2}. \tag{2.22}$$

Differentiating in respect to a results in

$$(1+2a')(a')_{.a}\lambda_r^2 = (1-2a), \tag{2.23}$$

and combining eq. (2.21) and eq. (2.23) results in

$$a' = \frac{1-3a}{4a-1}. \tag{2.24}$$

Also using eq. (2.22) in eq. (2.19) yields

$$16a^3 - 12a^2 + 3(3-\lambda_r^2)a - (1-\lambda_r^2) = 0, \tag{2.25}$$

which according to Brasil Jr. (2019) and Maalawi (2001) has the solution

$$\begin{cases} a = \frac{1}{2}[1-\lambda_r\cos\theta^+ - \cos\theta^-] \\ \lambda_r = \sqrt{(1+\lambda_r)} \\ \theta^\pm = \frac{1}{3}\cos^{-1}(\pm\lambda_r^{-1}). \end{cases} \tag{2.26}$$

The main advantage of this solution is that it is algebraic. Traditional BEMM as the one presented by Hansen (2015) used of iterative methods in order to interpolate the solutions for a and a', and needs to go through the calculation of the normal force coefficient C_n and the tangential force coefficient C_t in order to obtain the solution for a and a'. This procedure is more time intensive as it requires iterations and the conversion is not always stable, therefore the algebraic solution is more desirable.

In BEMT, the the blade is divided into N_{SEC} individual elements, where the thrust and momentum may be calculated for each element, and integrated over the radius. The thrust and power coefficients may be written as

$$dT_2 = \frac{1}{2}\rho N_B c \frac{u_0^2(1-a)^2}{\sin^2\phi} C_n dr \tag{2.27}$$

$$dM_2 = \frac{1}{2}\rho N_B c \frac{u_0^2(1-a)\omega r(1+a')}{\sin\phi\cos\phi} C_t r dr, \tag{2.28}$$

where N_B is the number of blades, c is the section's chord length, C_n is the normal and C_t the tangential coefficient, components of the total force given by

$$C_n = C_L \cos\phi + C_D \sin\phi \tag{2.29}$$

$$C_t = C_L \sin\phi - C_D \cos\phi. \tag{2.30}$$

According to Brasil Jr. (2019), with this relationships, it is possible to express the chord length as a function of the normal coefficient, which may be written as

$$c(r) = \frac{8\pi}{N_B}\frac{a}{(1-a)}\frac{\sin^2\phi}{C_n}r. \tag{2.31}$$

Additionally, in Figure (2.7) it can be observed that the twist angle θ is given by the difference between the inflow angle ϕ and the angle of attack α, that is

$$\theta = \phi - \alpha. \tag{2.32}$$

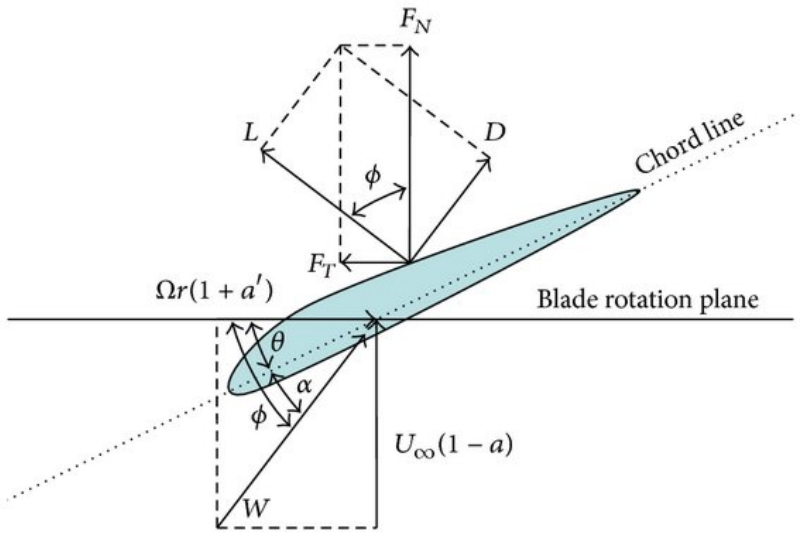


Figure 2.7 – Velocity triangle with the induction factors, and angles of pitch, attack, and effective angle of attack in the profile section (JOHNSON; GU; GAUNT, 2016).

2.2 Modeling Equations

The modeling equations for the motion of a fluid are the Navier-Stokes Equations may be written in integral form for a Newtonian isothermal fluid as

$$\begin{cases} \frac{\partial}{\partial t}\int_V \rho dV + \int_S (\rho\mathbf{u})\cdot\mathbf{n}dS = 0 \\ \frac{\partial}{\partial t}\int_V \rho\mathbf{u}dV + \int_S (\rho\mathbf{u}\mathbf{u})\cdot\mathbf{n}dS = -\int_S p\cdot\mathbf{n}dS + \int_S \mathbf{n}\cdot\bar{\bar{\tau}}dS,\ \bar{\bar{\tau}} = \mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right) \\ \int_V \rho\left(\frac{\partial e}{\partial t} + \mathbf{u}\cdot\nabla e\right)dV = -\int_V \nabla\cdot\mathbf{q}\,dV + \int_V \nabla\cdot(\bar{\bar{\tau}}\cdot\mathbf{u})\,dV + \int_V \nabla\cdot(p\mathbf{u})\,dV + \int_V \rho\mathbf{g}\cdot\mathbf{u}\,dV \end{cases} \tag{2.33}$$

where $\mathbf{u}$ is the velocity vector, p is the pressure, ρ is the specific mass of the fluid, μ is the dynamic viscosity, τ is the shear stress tensor (VERSTEEG; MALALASEKERA, 2007).

With the use of the Gauss theorem, the surface integrals may be transformed into a volume integrals resulting in the general differential form of the Navier-Stokes equations.

$$\begin{cases} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \\ \frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla \cdot \bar{\bar{\tau}} \\ \frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho e \mathbf{u}) = -\nabla \cdot \mathbf{q} \nabla \cdot (\bar{\bar{\tau}} \cdot \mathbf{u}) \nabla \cdot (p\mathbf{u}) + \rho \mathbf{g} \cdot \mathbf{u} \end{cases} \tag{2.34}$$

These are a highly complex set of equations with no general analytical solution for the time being, therefore needing to resort to numerical methods for the solution of the equations. As with any kind of mathematical model, it can be simplified to make the computational work easier, and in the case of the flow in a wind turbine, the simplification hypothesis made are:

1. **Incompressible Flow:** The flow displays negligible change in specific mass, this can be justified by the low Mach number of the flow.
2. **Isothermal Flow:** The flow displays negligible change in temperature, therefore the energy equation is not considered.
3. **Unsteady Flow:** The flow behaviour is time dependent
4. **Newtonian FLuid:** Air is considered a Newtonian fluid, and with no change in temperature, the viscosity is considered constant.
5. **Laminar Steady Flow Upstream:** The flow is considered laminar upstream as for an ideal representation of the flow, and a better representation of vortex structures in the wake.[2]

The Navier-Stokes equations may then be written in index notation as

$$\frac{\partial}{\partial x_j} u_j = 0, \tag{2.35}$$

for the continuity equation, and

$$\frac{\partial}{\partial t} u_i + u_j \frac{\partial}{\partial x_j} u_i = -\frac{1}{\rho} \frac{\partial}{\partial x_i} p + \nu \frac{\partial^2}{\partial x_j \partial x_j} u_i, \tag{2.36}$$

for the momentum equation, where $\nu = \mu/\rho$ is the kinematic viscosity.

[2] According to (VERMEER; SØRENSEN; CRESPO, 2003) this hypothesis is not a valid representation of the flow at the operating site, because wind changes in both intensity and direction not to mention the changes in turbulence intensity for innumerable reasons.

These equations are not optimal for numerical solutions, because they are not dimensionless, therefore it is recommended to perform a non-dimensionalization of the equations. Using characteristic values for the variables, the equation (2.35) continues pretty much the same, while equation (2.36) becomes

$$\frac{\partial}{\partial t}u_i + u_j\frac{\partial}{\partial x_j}u_i = -\frac{\partial}{\partial x_i}p + \frac{1}{Re}\frac{\partial^2}{\partial x_j \partial x_j}u_i, \tag{2.37}$$

where Re is the Reynolds number, which is defined as $Re = V_0 L/\nu$, where V_0 is the characteristic velocity, and L is the characteristic length. The Reynolds number is a dimensionless number that represents the ratio between inertial forces to viscous forces. Because of the geometrical similarity principle, the Reynolds number is a good indicator of the flow behavior.

While the Navier-Stokes can be directly solved numerically, a method called DNS (Direct Numerical Simulation), it is not the most efficient way of solving the equations, because it is computationally expensive. This expensiveness can be illustrated by considering the case of turbulent flow. In turbulence flow, eddies can be found in a wide range of scales, according to Kolmogorov's theory of turbulence (KOLMOGOROV, 1991), with great discrepancy between the smallest scales, and the largest scales.

Because the smallest eddies are one of the most important structures in the turbulent flow, responsible for the dissipation of turbulent kinetic energy, it is necessary to solve them accurately, with a mesh size smaller than the eddy size. It can be easily thought that for a fluid domain as in the case of a wind turbine, with a radius of say $50m$, and therefore a $500m$ far wake, this computation is not feasible. Therefore it is necessary to resort to turbulence models to solve the equations practically.

2.3 Turbulence Model

Turbulence models are mathematical models that are used to predict the effects of turbulence in a fluid flow. They are used in conjunction with the Navier-Stokes equations to solve the equations practically.

When it comes to wind turbines turbulence models are classified into:

1. **Reynolds Averaged Navier-Stokes (RANS) Models:** These models are based on the Reynolds decomposition, which is a mathematical tool used to separate the flow into a mean and a fluctuating part of the flow.

2. **Unsteady Reynolds Averaged Navier-Stokes (URANS) Models:** These models are also based on the Reynolds decomposition, but they are time dependant, and therefore they are more accurate than RANS models.

3. **Large Eddy Simulation (LES) Models:** These models are based on the decomposition of the flow into large and small eddies, and the large eddies are solved directly, while the small eddies are modeled.

4. **Detached Eddy Simulation (DES) Models:** These models are based on the LES models, but they are used in the near-wall region, where the LES models are not accurate.

2.3.1 RANS Model

The RANS models are based on the Reynolds decomposition, which is performed by:

$$\phi = \langle \phi \rangle + \phi', \tag{2.38}$$

where ϕ is any property of the flow, $\langle \phi \rangle$ is the mean value of the property, and ϕ' is the fluctuating part of the property. The mean value is defined as

$$\langle \phi \rangle = \frac{1}{T} \int_0^T \phi dt, \tag{2.39}$$

where T is the period in which the mean value is calculated.

The most used RANS models are based on three turbulent properties, namely, the turbulent kinetic energy k, the turbulent dissipation rate ϵ, and ω.

The turbulent kinetic energy is defined as

$$k = \frac{1}{2} \langle u_i' u_i' \rangle, \tag{2.40}$$

where u_i' is the fluctuating part of the velocity vector and the turbulent dissipation rate is defined as

$$\epsilon = \nu \left\langle \frac{\partial u_i'}{\partial x_j} \frac{\partial u_i'}{\partial x_j} \right\rangle, \tag{2.41}$$

where ν is the kinematic viscosity, and $\partial u_i'/\partial x_j$ is the rate of strain tensor.

The ω equation is a transport equation for the specific dissipation rate ω, which is defined as

$$\omega = \frac{\epsilon}{k}. \tag{2.42}$$

Applying the Reynolds decomposition to the Navier-Stokes equations and taking the time average results in the RANS equations, which are given by

$$\frac{\partial}{\partial x_j} \langle u_j \rangle = 0; \frac{\partial}{\partial x_j} u_j' = 0 \tag{2.43}$$

for the continuity equation, and

$$\langle u_j \rangle \frac{\partial \langle u_i \rangle}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \langle p \rangle}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\nu \frac{\partial \langle u_i \rangle}{\partial x_j} - \langle u_i' u_j' \rangle \right). \tag{2.44}$$

With the previous manipulations, the RANS equations are now in a form that has six new unknowns, because $\overline{u_i' u_j'}$ is the Reynolds stress tensor, which is a symmetric tensor, is given by

$$\overline{u_i' u_j'} = \overline{\begin{bmatrix} u_1' u_1' & u_1' u_2' & u_1' u_3' \\ u_2' u_1' & u_2' u_2' & u_2' u_3' \\ u_3' u_1' & u_3' u_2' & u_3' u_3' \end{bmatrix}}, \tag{2.45}$$

there are still however only 4 equations, which cause the so-called Closure Problem. And to deal with this problem more equations are added to the system to model turbulent terms. To model the Reynolds stress tensor the Boussinesque Hypothesis is applied, which consists in treating turbulence much like viscous forces, since both are of dissipation nature. Therefore, the Reynolds Stress Tensor is modeled in terms of a turbulent viscosity μ_T as

$$\overline{u_i' u_j'} = \nu_T \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij}, \tag{2.46}$$

2.3.2 K-Omega SST Model

The $k - \varepsilon$ model was proposed by Laudner (1973) working very well in free-flow conditions, however, it performs poorly near the walls of the domain and relies on wall functions to approximate the influence of the boundary layer. The $k - \omega$ model by Wilcox, on the other hand, performs well near walls and has poor performance in free stream conditions due to an over-reliance on boundary conditions. The $k - \omega$SST model tries to get the best of both worlds and uses the $k - \varepsilon$ equation in ω form for the main flow, and the $k - \omega$ near the wall (MENTER, 1994). Currently, this approach yields the most accurate results for flows over turbines.

2.4 Simulation Methods

There are three main methods of simulating the rotation of a wind turbine:

1. **Actuator Line Method (ALM):** This method consists of modelling the rotor blades as forces acting on the flow, and therefore the blades are modeled through tabulated data and the flow is solved in a stationary reference frame. This allows for great computational efficiency and yields good preliminary results (SORENSEN; SHEN, 2002), it is not, however, a good representation of the flow, since the blades are not modeled, and for instance stall conditions due to wrong geometries may not be captured.

2. **Moving Reference Frame (MRF):** This method consists of basically inverting the frame of reference, that is making the rotor stationary while the fluid rotates around it. This method is solved using steady-state equations which means that time-dependent

effects are not captured. This method is more computationally expensive than the ALM, however, it can capture integral properties of the flow, with some limitations, such as the wake meandering, and the tip vortices (MEHDIPOUR, 2014).

3. **Sliding Mesh Method (SMM):** This method consists of modelling the rotor in a rotating domain, and therefore the flow is solved in a stationary reference frame, and with each iteration the mesh of the rotating domain has its positions recalculated, and the properties in the interface between domains are updated through interpolation. This method is the most computationally expensive, due to the rotation of the mesh, and the interpolation of properties on interfaces, however, it is the most physically accurate method of the three.

It can be said that the MRF method is a middle ground between the ALM and the SMM, not only in computational cost but also in accuracy. The MRF method gives a good idea of the flow and has the advantage of the easiest implementation (MEHDIPOUR, 2014).

2.4.1 MRF Formulation

The MRF method consists of changing reference frames, which means that the rotor is kept stationary while the fluid rotates around it. Let $\vec{r}$ be the coordinate of a point P in the stationary coordinate system K, and $\vec{R}$ is the position of coordinate of the center of the moving coordinate system K', and $\vec{r'}$ is the distance from a given point to the center of K', then it can be said that

$$\vec{r} = \vec{R} + \vec{r'}.$$

Therefore velocity of the point $\vec{r}$ is given by

$$\vec{u} = \frac{d\vec{x}}{dt} = \vec{u_r} + \vec{\omega} \times \vec{r'},$$

and after taking the time derivative of the previous equation yields

$$\frac{d\vec{u}}{dt} = \frac{d'\vec{u_r}}{dt} + \frac{d'\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u_r} + \omega \times \omega \times \vec{r}.$$

Considering steady state, however,

$$\frac{d\vec{u}}{dt} = \nabla \cdot (\vec{u_r}\vec{u_r}) + 2\vec{\omega} \times \vec{u_r} + \omega \times \omega \times \vec{r},$$

which is naturally the left side of the Navier-Stokes equations, therefore the MRF formulation of the Navier-Stokes equations is given by

$$\nabla \cdot (\vec{u_r}\vec{u_r}) = -\frac{1}{\rho}\nabla p + \nabla \cdot (\nu \nabla \vec{u_r}) - 2\vec{\omega} \times \vec{u_r} - \omega \times \omega \times \vec{r}, \tag{2.47}$$

where the term $-2\vec{\omega} \times \vec{u_r}$ models the Coriolis force, and the term $-\omega \times \omega \times \vec{r}$ models the centrifugal force.

3 Geometry Generation

The geometry is based in a toroidal propeller rotor presented in the patent (SEBASTIAN; STREM, 2019), consists of three inclined circular rings, with their centers positioned at $2\pi/3$ radians appart, and $R/2$ far from the center. This geometry does not seem to have any aerodynamic profile in its cross sections. The pitch angle (inclination angle) is not disclosed in the patent, however, it can be estimated to be around 20° to 30°. The main hypothesis of this work is to investigate if this innovative design is capable of being used (with adaptations provided using BEMT) as a turbine, and for that reason, the overall design aspects were preserved.

As it can be seen from Figure (3.8), the highest power coefficients were obtained for the four-bladed rotor. Additionally, the four-bladed rotor presents a geometric simplification, when it comes to the geometric description of the toroidal blades, which will be discussed further in this chapter. For these reasons, the four-bladed rotor was chosen as the base geometry for this work.

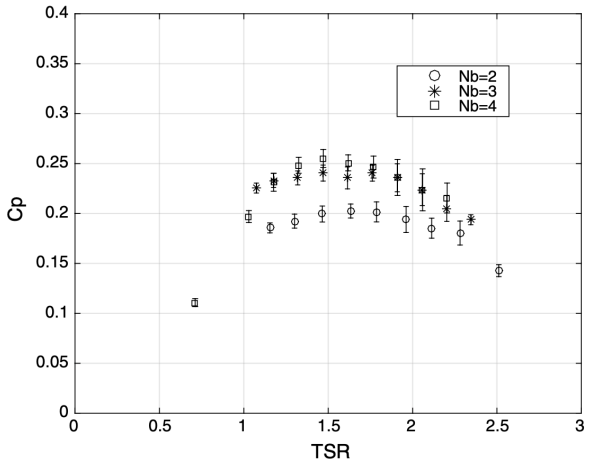


Figure 3.8 – Wind tunnel results for the Power Coefficient c_p for different number of blades presented by (BRASIL JUNIOR et al., 2019).

The software for both the BEM algorithm and the geometry generation was written in Python 3.9.5, due to its good prototyping capabilities and high level of abstraction. The BEM algorithm was written using both functional and object-oriented programming (OOP) paradigms, because of the ease of access to information, the aerofoil description was written using OOP, due to the way easier manipulation of the coordinates, and the remaining module was written using functional and imperative programming.

The standard way of drawing a blade is to use a skeleton straight line, which can be placed in either the leading edge or the aerodynamic center of the profile which in NACA

profiles is placed in $1/4$ of the chord length. And then stacking sections over one another separated by the spatial step dr used in the skeleton line. This approach may be used as well in curved blades, however, the profiles need to be rotated so that the normal angle of the profile's surface is always pointing in the same direction as the tangent vector of the curve.

The geometry generation process can thus summarized in the following steps:

1. Generate the blade curve;
2. Run BEM;
   Run Xfoil;
   Get optimal angle of attack;
   Run Algorithm 1;
   Run Algorithm 2;
   Calculate mean properties;
   Check Reynolds number;
3. Load the profiles;
4. Rotate the profiles to the angle of attack;
5. Scale the profiles to the chord length;
6. Rotate the profiles to the curve's angle;
7. Translate the profiles to the correct position in the curve;
8. Rotate the last profiles to match the middle profiles of the second blade;
9. Reflect the profiles in the x and y axis;
10. Rotate and copy the blades by $\pi/2$ radians in respect to the center of the coordinate system as it is the center of the rotor;
11. Remove the first profile from the array because has zero length;

3.1 Geometry of the Toroidal Rotor

The overall geometry of the toroidal rotor is presented in Figure (3.9). It is composed of four circumference all of them with an intersection point and centers placed in a circular pattern $2\pi/N_B$ $(\pi/2)$ apart from each other. For this disposition to work, that is for the rotor to be of the desired diameter, the blade circles must have a radius of $R/2$, for any number of

blades. It is also relevant to point out that the angular span of a four-bladed rotor must be $3\pi/2$ geometrically.

Lastly, when creating an array for the circular blades with angles ($\beta = [0,3\pi/2]$), the corresponding radial positions are not evenly distributed in a line, because if the curves are described by

$$\begin{cases} x_i = 0_i \\ y_i = \frac{R}{2}\cos(\beta_i) - \frac{R}{2} \\ z_i = \frac{R}{2}\sin(\beta_i) \end{cases}, \tag{3.1}$$

then the radial positions will be given by

$$r_i = \sin(\beta_i), \beta_i = [0, \pi],$$

which is not coherent with the rotor coordinate system, where the radial positions should be given by

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

Figure (3.9) illustrates the geometry of the guiding curves of the toroidal rotor.

3.2 Engineering Parameters

In order to generate a rotor, the working conditions of it must be considered. The engineering parameters therefore are

- `NACA`: the code for the four-digit NACA profile chosen [1];
- `NSEC` - N_S: number of profile sections to construct a straight-bladed rotor;
- `OMEGA` - ω: rotational speed of the rotor;
- V_0: velocity of the flow far upstream;
- R: radius of the rotor;

[1] For the current implementation, these profiles must be symmetric (00XY), because in the transition from the first two-thirds of the curved blade to the last third, a change of orientation of the profile happens so that the lift has the right orientation, which means that the lower surface of the last profile in the main blade encounters the upper surface of the first from the last part of the blade.

Although it is possible to use asymmetric profiles, there is a need to implement a smooth transition from the base asymmetric profile to the top symmetric profile, this smooth transition may be done by placing an asymmetric profile in the base, 4415 for instance, and a symmetrical profile (0015) on the outer radius position, and based on the number of sections, calculate the coordinates of the in-between profiles by tracing lines between the same index points from both base profiles.

The issue with this method is that each profile in the smooth blade must have its lift and drag coefficient tables calculated individually because each profile has a unique and non-standardized shape.

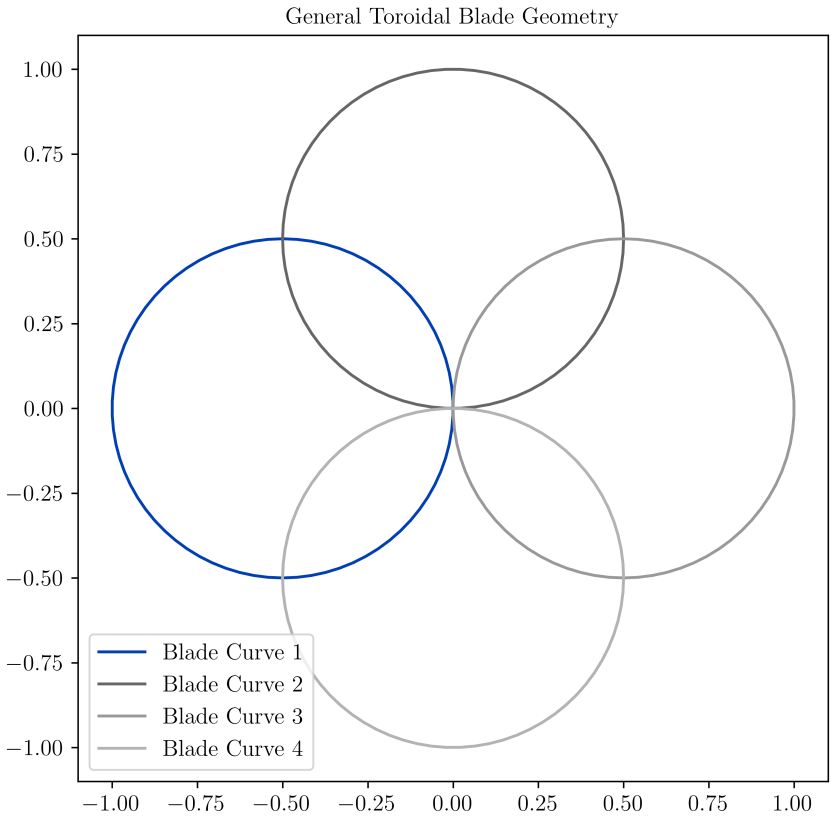


Figure 3.9 – Geometry of the toroidal rotor with four blades

- R_H: radius of the turbine's hub;
- `N-B` - N_B: number of blades in the rotor;
- `RHO` - ρ: specific mass of the fluid in which the turbine will operate;
- `MU` - μ: absolute viscosity of the fluid;
- `REYNOLDS` - Re: Reynolds number of the profiles. This Reynolds number is calculated through

$$Re = \frac{V_0 c}{\nu}, \nu = \frac{\mu}{\rho},$$

with

$$V_0 = \sqrt{[u_0(1-a)]^2 + [\omega R(1+a')]^2},$$

where c is the characteristic chord length of the profiles. This Reynolds number does not need to be exact rather the order of magnitude must be the same, for instance, $Re = 1.0 \cdot 10^5$ and $Re = 2.5 \cdot 10^5$ will yield practically similar results, which means that the input may be an approximation.

The algorithm inputs variables and their respective values, units, and types may be observed in Table (3.1).

Property	**Symbol**	**Rotor I**	**Rotor II**	**Rotor III**	**Unit**
Profile NACA code	–	“0015”	“0015’	“0015’	–
Number of Sections	N_S	200	100	100	–
Tip Speed Ratio	ω	4.0	4.0	4.0	–
Free Stream Velocity	V_0	8.0	8.0	8.0	m/s
Radius of the Rotor	R	0.145	0.11	1.1	m
Scale	–	1:1	1:1	10:1	–
Radius of the Hub	R_H	0.0336	0.015	0.015	m
Number of Blades	N_B	4	4	4	–
Density	ρ	1.225	1.225	1.225	kg/m^3
Dynamic Viscosity	μ	$1.849e-5$	$1.849e-5$	$1.849e-5$	Ns/m^2
Kinematic Viscosity	ν	$1.560e-5$	$1.560e-5$	$1.560e-5$	m^2/s
Reynolds Number	Re	$6.7e+4$	$3.8e+05$	$3.8e+06$	–

Table 3.1 – Algorithm input variables, their respective mathematical symbol, stored value, unit and type

3.3 Generating the Curvature of the Blade

The first step in generating the curved blade is to generate the blade curve, because each position in the curve will be used as the aerodynamic center of a profile. The generation of the curve was done similarly to the description in Section (3.1), that is, by generating a circular curve with

$$C_{k,i} = \begin{cases} x_k = x_{k,0} = 0_k \\ y_k = x_{k,1} = \frac{R}{2}\cos(\gamma_k) - \frac{R}{2} \\ z_k = x_{k,2} = \frac{R}{2}\sin(\gamma_k) \end{cases} \text{, for all } k\text{in}[0,N_{\text{SECT}}) \cap \mathbb{N}, \tag{3.2}$$

where $N_{\text{SECT}} = 1.5 \cdot N_{\text{SEC}}$ is the total number of sections in the whole blade, and γ being the angle of the 2D polar coordinate system (r, γ).

As explained in Section 3.1, this ought to be done as the first step, because BEM requires the radial positions.

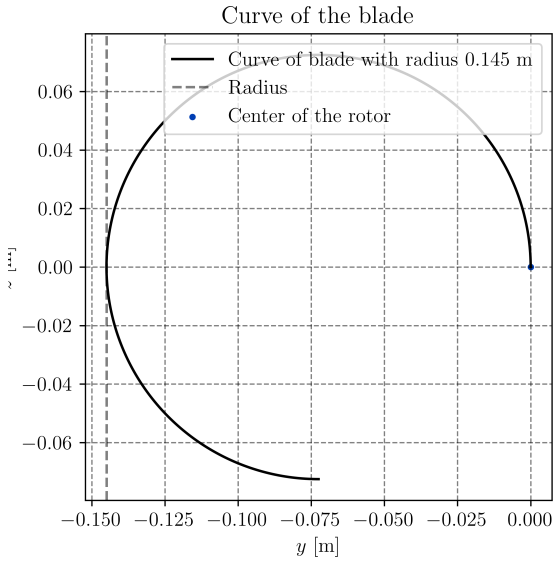


Figure 3.10 – Blade curve for a 4 (four) bladed toroidal rotor

Code 3.1 – Curve Generation Algorithm Section

```
# NSEC = 100
# NSECT = int(1.5*NSEC)
# N_B_STRAIGHT = 4

beta = np.linspace(0, 3*np.pi/2, NSECT)

curve = np.array([np.zeros(NSECT),
                  np.cos(beta)-1,
                  np.sin(beta)])*(R/2)

curve = np.transpose(curve)
```

3.4 Implementation of BEM

3.4.1 XFOIL implementation

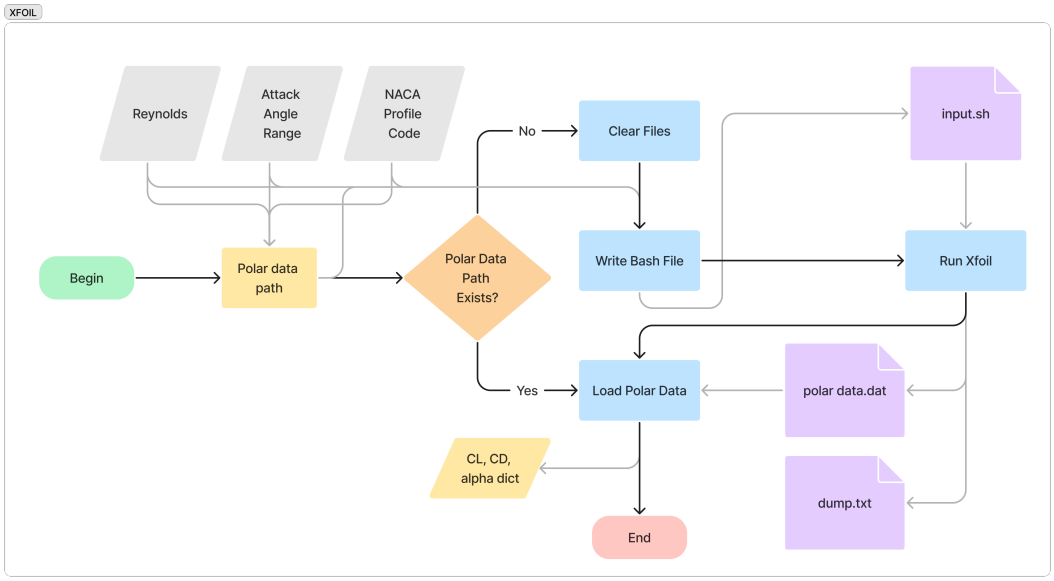


Figure 3.11 – XFOIL Flow-chart - `xfoil.py`

XFOIL is a program developed by Mark Drela and Harold Youngren at MIT Aeronautical Engineering Department. It is a program that uses a panel method to calculate the lift and drag coefficients of a given profile. It is written in FORTRAN 77 with plotting functions written in C and it is available openly for download. The program runs in a terminal using a command line interface.

Running XFOIL with the aid of a Python script allows for centralized control of the blade generating method, and for easy change in any relevant parameter, which are: the code for the four-digit **NACA** profile chosen; **Reynolds number** of the profiles, using the relative velocity $V_{\mathrm{REL},1}$ as the characteristic velocity, the mean chord c as the characteristic length and the kinematic viscosity ν for air; the **initial angle of attack** α_i of the profile; the **final angle of attack** α_f of the profile;the **angle of attack's step** $\Delta\alpha$. Figure (3.11) illustrates the workings of the XFOIL automation algorithm implemented in this work.

Figure (3.12) shows the polar data results obtained from XFOIL for the NACA 0015 profile using the Reynolds number $Re = 5.4 \cdot 10^4$, and the angle of attack range $\alpha = [0, 20]$ with a step of $\Delta\alpha = 0.025$. The results are in accordance with the expected results for a NACA 0015 profile, which means that the XFOIL automation algorithm is working properly.

3.4.2 BEM Method

The BEM algorithm was implemented in accordance with the algorithm from the work of Brasil Jr.,A (2019), and the pseudo-algorithm for this implementation is presented in

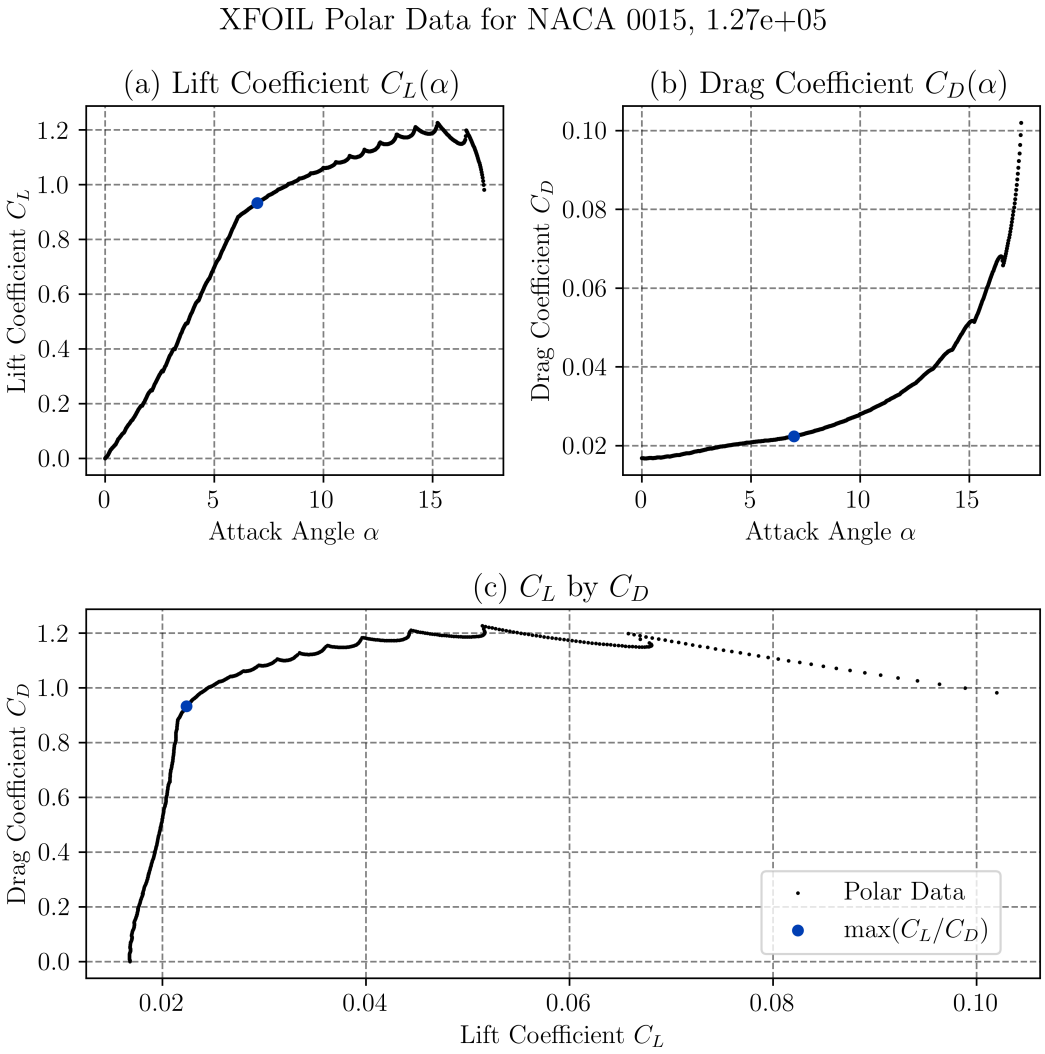


Figure 3.12 – Polar data results for NACA 0015 profile obtained using XFOIL. (a) Lift coefficient C_L vs Angle of Attack α; (b) Drag coefficient C_D vs Angle of Attack α; (c) Lift coefficient C_L vs Drag coefficient C_D

Algorithm (1), and Algorithm (2). The BEM implementation was done using OOP because it allows for an easier access to the information.

The first algorithm takes as inputs the upstream velocity u_0, the angular velocity ω and the radial positions of the profiles, and returns the axial and tangential induction factors, a and a', respectively, as well as the inflow angle ϕ.

Secondly, BEM uses a function (`get-optimal()`) to go through the polar data obtained from XFOIL and find the optimal angle of attack α_o that generates the optimal lift and drag coefficients C_{Lo} and C_{Do}, respectively. This process is done by finding the index, where the lift-to-drag ratio C_L/C_D is maximum. The process of finding the optimal properties from the polar data is done this way instead of calculating the derivative of the lift curve and finding the zero, because as seen in Figure (3.12), the maximum lift coefficient will also result in high drag coefficients, by using the ratio, the result will always be the maximum lift

Algorithm 1 BEM Algorithm 1 (BRASIL JUNIOR et al., 2019)

Require: $\{u_0, \omega, r_i\}$

for $(i = 0; i < N_{SEC};\ i \leftarrow i + 1)$ **do**

$\Lambda_{r,i} \leftarrow \frac{\omega r_i}{V_0}$

$\lambda_{r,i} \leftarrow \sqrt{1 + \Lambda_{r,i}^2}$

$\theta^+ \leftarrow \frac{1}{3}\arccos(1/\lambda_{r,i})$

$\theta^- \leftarrow \frac{1}{3}\arccos(-1/\lambda_{r,i})$

$a_i \leftarrow \frac{1}{2}\left[1 - \lambda_{r,i}(\cos\theta^+ - \cos\theta^-)\right]$

$a_i' \leftarrow \frac{(1-a)}{(4a-1)}$

if $\Lambda_{r,i} = 0$ **then**

$\Lambda_{r,i} \leftarrow 10^{-5}$

end if

$\phi_i \leftarrow \arctan\frac{(1-a_i)}{\Lambda_{r,i}(1+a_i')}$

end for

return $\{a, a', \phi\}$

to the minimum drag.

The third step is to calculate the chord c and twist angle θ of the profiles. This is done through Algorithm (2), which receives as inputs the axial induction factor a, the inflow angle ϕ, the radial positions r_i, the number of blades N_B, the optimal angle of attack α_o, the optimal lift coefficient C_{Lo}, the optimal drag coefficient C_{Do}, and a boolean variable `tip-correction`, which defaults to `False`, because in toroidal blades there is no tip, therefore Prandtl's tip loss factor is not applicable. The algorithm returns the chord c_i and the twist angle θ_i of each profile as a dictionary. The algorithm is implemented in Python and can be found in Appendix (A.3).

In many cases, it is possible to enter a wrong estimate for the Reynolds number, which would lead all results from BEM to be invalid. For this reason, the mean properties from all the profiles are calculated, that is $\bar{a}$, $\bar{a}'$, $\bar{c}$ and $\bar{\theta}$, and then the Reynolds number is calculated for these mean properties as

$$Re_{\text{new}} = \frac{\bar{c}V_{\text{REL},1}}{\nu}, V_{\text{REL},1} = \sqrt{\left[u_0(1-\bar{a})\right]^2 + \left[\omega R(1+\bar{a}')\right]^2}. \tag{3.3}$$

Algorithm 2 BEM Algorithm 2 (BRASIL JUNIOR et al., 2019)

Require: $\{a_i, \phi_i, r_i, N_B, \alpha_o, C_{Lo}, C_{Do}, \texttt{bool tip-correction}\}$

$\theta_i \leftarrow 0_i; c_i \leftarrow 0_i$

$R \leftarrow \max(r_i)$

for $i = 0; i < N_{SEC}; i \leftarrow i + 1$ **do**
  $\theta_i \leftarrow \phi_i - \alpha_o$
  if $\theta_i < 0$ **then**
    $\theta_i \leftarrow 0$
  end if

  $C_n \leftarrow C_{Lo} \cos\phi_i + C_{Do} \sin\phi_i$

  $c_i \leftarrow \frac{8\pi r_i}{N_B} \frac{a_i}{(1-a_i)} \frac{(\sin\phi_i)^2}{C_n}$

  if `tip-correction` is `True` **then**

    $f_{tip} \leftarrow \frac{N_B[1-(r_i/R)]}{2(r_i/R)\sin\phi_i}$

    $F_{tip} \leftarrow \frac{2}{\pi} \arccos\left(e^{-f_{tip}}\right)$

    $f_{hub} \leftarrow \frac{N_B(1-R_{hub})}{2r_i \sin\phi_i}$

    $F_{hub} \leftarrow \frac{2}{\pi} \arccos\left(e^{-f_{hub}}\right)$

    $F \leftarrow F_{tip} \cdot F_{hub}$

    $c_i \leftarrow c_i \cdot F$
  end if
end forreturn $\{c_i, \theta_i\}$

If the absolute error from the new Reynolds number and the old Reynolds number

$$E_{\text{Re}} = \frac{|Re_{\text{new}} - Re_{\text{old}}|}{Re_{\text{old}}} \tag{3.4}$$

is greater than 0.1, the XFOIL polar data is recalculated using the new Reynolds number and the process is repeated until the absolute error is less than 0.1. This ensures that the input Reynolds number is of the same order of magnitude as the Reynolds number of the profiles.

Lastly, the results from BEM are saved into a file, so that they can be used in the future, or for further analysis.

3.5 Geometry Validation

In order to validate the BEM algorithm, a straight blade rotor was generated based on the HK-10 (BRASIL JUNIOR et al., 2019) (MENDES et al., 2020) rotor, which has the engineering parameters presented in Table (3.2).

Parameter	Symbol	Value
Number of Blades	N_B	4
Number of Sections	N_{SEC}	15
Tip Speed Ratio	TSR	1.6
Reynolds Number	Re	$1.85 \cdot 10^5$
Free Stream Velocity	V_0	$2.5\ m/s$
Radius of the Rotor	R	$1.1\ m$
Radius of the Hub	R_{HUB}	$0.15\ m$
Kinematic Viscosity	ν	$1.0035 \cdot 10^{-}6\ m^2/s$

Table 3.2 – HK-10 Project Parameters

The validation of the BEM algorithm was done by plotting the blade geometry and the twist angle θ by the adimensional radius r/R, as seen in figure (3.13). Subsequently, the relative error between the HK-10 chord and twist angles was calculated, and the results can be seen in Figure (3.14).

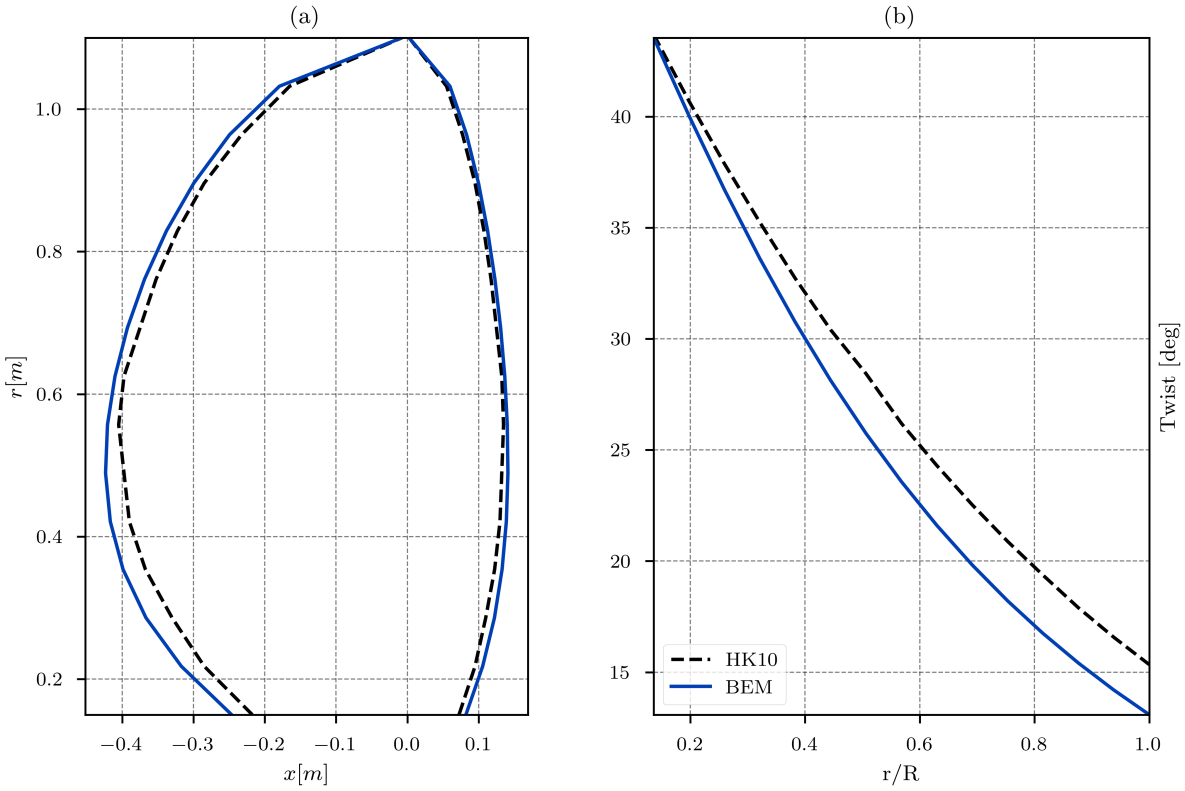


Figure 3.13 – Comparisson between the Hk-10 geometry and the geometry generated by BEM

In order to calculate the error the difference between chord lengths were normalized by the radius of the rotor R, due to the fact that a 0.01 m error in the chord length is much

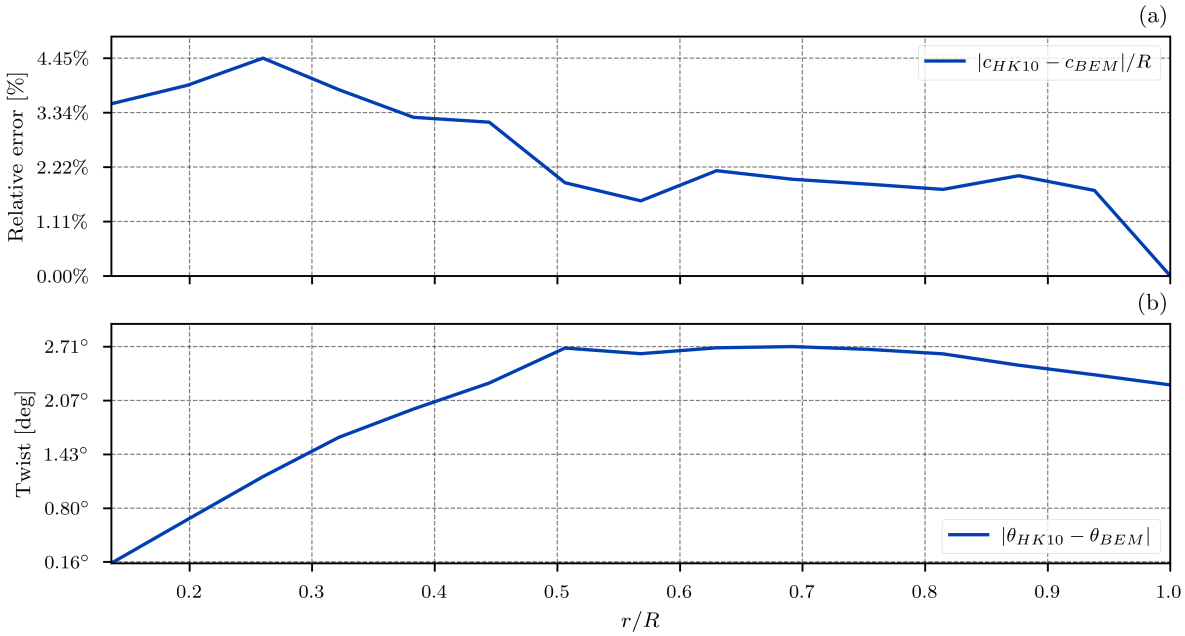


Figure 3.14 – Error between the Hk-10 geometry and the geometry generated by BEM

more significant as the rotor radius decreases. The twist angle error was calculated by simply taking the difference between the twist angles.

3.5.1 Conclusions

During the validation process, it was observed that although the chord lengths are not sensitive to the Reynolds number, an increase or decrease in the Reynolds number with a second order of magnitude while calculating the twist angle, could result in a 24º difference in the twist angle, since XFoil calculates the optimal attack angle from 0º to about 12.5º, it can be said that this error can indeed be significant.

3.6 Rotor I

The first rotor geometry is subdivided into two distinct sections. The first section goes from the angles $\beta = [0,\pi]$, that is from the center of the coordinate system to the blade's radius. The second section goes from $\beta = [\pi, 3\pi/2]$, that is from the blade's radius to the middle of another blade , as illustrated in Figure (3.9). The reason for this division is that these two sections are anti-simetric, and therefore require different treatment.

3.6.1 First Part of the Blade $[0,\pi]$

First of all an array with the coordinates of every profile was created using the NACA 0015 coordinates, therefore the shape of this array is $(N_{SEC}, N, 3)$, where N_{SEC} is the number of sections in the first part of the blade, N is the total number of points in the profile, and 3

is the number of dimensions of the rotor. It is important to note that this array is constructed in such a way that all the coordinates in z are 0.

The coordinate system $\mathbb{R}^3$ is oriented in such a way that the profile's chord is parallel to the x axis, $c_{j,i} \parallel e_0, \forall j \in [0,(N/2)-1] \cap \mathbb{N}$.

Then, all the profiles are moved so that the center of the coordinate system $\mathbb{R}^3$ is at the aerodynamic center of the profiles. Let $P_{k,j,i}$ be an array with the coordinates as previously described, then this translation operation is simply

$$P_{k,j,0} \leftarrow P_{k,j,0} - \frac{1}{4}, \tag{3.5}$$

because the chord is still unity $c_k = 1.0$ from the NACA profile generation algorithm[2].

Thirdly, the profiles are scaled to the desired chord length c_i, which is done by the scaling operation

$$P_{k,j,i} \leftarrow P_{k,j,i} \cdot c_k. \tag{3.6}$$

Next, all the profiles in $P_{k,j,i}$ are rotated in relation to the z axis, giving the blade its twist. This is done by the rotation matrix

$$P_{k,j,i} \leftarrow Pj,k,i \cdot R_z(-\theta_k), \tag{3.7}$$

where $R_z(\theta_k)$ is the rotation matrix around the z axis by the angle θ_k.

With the blade profiles twisted properly, the next step is to rotate the profiles in such a way that the normal vector of the profile's surface is always pointing in the same direction as the tangent vector of the curve, that is $n_{k,j,i} \parallel \tau_{k,i}$. For this condition to be satisfied, the y coordinate from the profiles must be parallel to the radius of the blade, so the profiles can just be rotated in the x axis by the angle $-\beta_k$ used to define the blade's curve , that is

$$P_{k,j,i} \leftarrow P_{k,j,i} \cdot R_x(-\beta_k). \tag{3.8}$$

The profiles were then reflected in the x and y axis so that the profiles are oriented in the correct direction.

$$P_{k,j,i} \leftarrow -P_{k,j,i}, i = 1,2 \tag{3.9}$$

After this, the profiles were translated into the corrected position.

$$P_{k,j,i} \leftarrow P_{k,j,i} + C_{k,i}, \tag{3.10}$$

Finally, the profiles were copied into a new array $B_{k,j,i}, \forall k \in [0,N_{SECT}) \cap \mathbb{N}$, $j \in [0,N) \cap \mathbb{N}, i = 1,2,3$, that is with shape $(N_{SECT}, N, 3)$, where N_{SECT} is the total number of sections. Therefore,

$$B_{k,j,i} \leftarrow P_{k,j,i}, \forall k \in [0,N_{SEC}) \cap \mathbb{N}. \tag{3.11}$$

[2] This algorithm was used rather than extracting the profiles from XFOIL, for the STL lofting algorithm, requires for a nonzero thickness trailing edge.

3.6.2 Second Part of the Blade $[\pi, 3\pi/2]$

For the second part of the blade, the profiles from the first part of the blade were copied to the blade's array $B_{k,j,i}$ in reverse order, that is

$$B_{m,j,i} \leftarrow P_{N_{SEC}-k-1,j,i}, \forall k \in [0,N_{SEC}) \cap \mathbb{N} \text{ and } m \in [N_{SEC}, N_{SECT}) \cap \mathbb{N}. \tag{3.12}$$

However, because the second part of the blade meets another blade, that Does blade 1 meet blade 2, because blade 2 has a twist angle in the position of the encounter, which means that the last profiles from blade 1 will cross the middle profiles of blade 2. In order to solve this problem a last operation must be made, which is to rotate the profiles in the second part of the blade in the z axis, so that the last profile of the second part of blade 1 meets the intermediary profiles at approximately their chords. However, this cannot be done only in the last profile, rather this change in angle must diffuse through the second part of the blade. Because this inclination of the last profiles must be greater than in the first, an exponential transition function was employed,

$$\gamma_k = -\theta_{(N_{SEC}/2)-1-k} \cdot D^{1-k}, \tag{3.13}$$

where D is an adjustment constant that controls the rate of change of the diffusion, in this work $D = 1.08$ was used because it was the value that produced the best result, this is therefore an empirical constant, which is not ideal.

It is important to note nevertheless, that the rotation operation is done in relation to the center of the coordinate system, hence to perform this rotation operation it is necessary to translate profiles to the center, rotate them and then translate them back to their original position. There was an issue however with the definition of the aerodynamic center of the profile, thus they were translated by $-R$, rotated and then translated back to their original position, which creates a slight undesired curvature in the intersection to the front of the rotor.

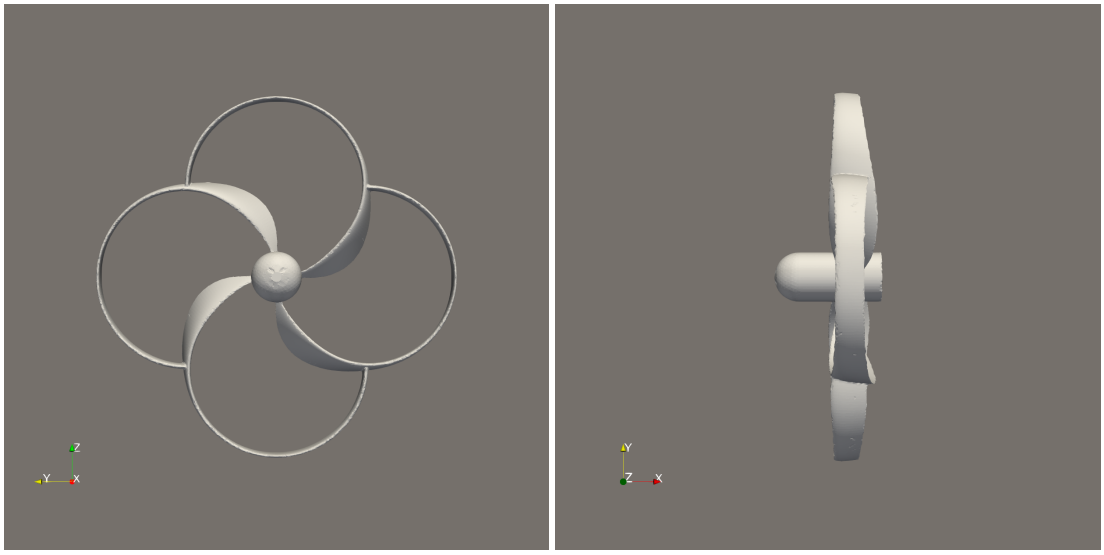
3.6.3 Generation of the Remaining Blades

The remaining blades were generated by a rotation of the first blade in the x axis by the angles $\sigma_m \in [0,\pi/2,\pi,3\pi/2], m = 1,2,3,4$, therefore generating the four blades of the rotor. The rotation was done by

$$W_{m,k,j,i} \leftarrow B_{k,j,i} \cdot R_x(\sigma_m), \tag{3.14}$$

where $W_{m,k,j,i}$ is the array with the coordinates of the m-th blade.

The results of all these geometric operations may be seen in Figure 3.15.



(a) Front view (b) Side View

Figure 3.15 – Rotor I front and side view images

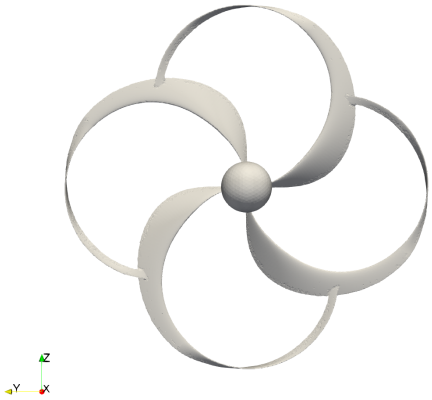
3.7 Rotor II and III

For the both rotor geometry, a different approach to the control of the position and angle of the profiles were used. Instead of using arrays with profile coordinates, and controlling them in batches, a class was created to represent the profiles, this allows for not only the storage of important properties, such as the angles used in each of the directions of the coordinate system and the position of the aerodynamic center of each profile, but it also allows for control of all parameters of a single profile individually, solving the previously mentioned problem of the curvature in the intersection. The implementation of this class can be found in Appendix (A.1).

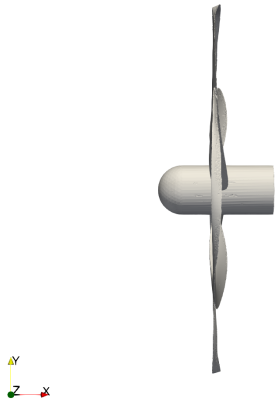
Both of the rotors were generated using essentially the same algorithm, and chord distribution, the only difference between the two rotors is the twist angle distribution. The chords were distributed in a way that for the initial, profiles the chord length was calculated using the BEM algorithm as for to implement the hub correction, this distribution goes up to the maximum chord length, after this section, if the chord length is smaller than half of the mean chord $\bar{c}/2$, than the chord length is set to $\bar{c}/2$. Naturally, this distribution goes up to the last profile. For the second section of the blade, the chord length is set to $\bar{c}/2$.

For the twist distribution, the blade was instead divided up into three sections:

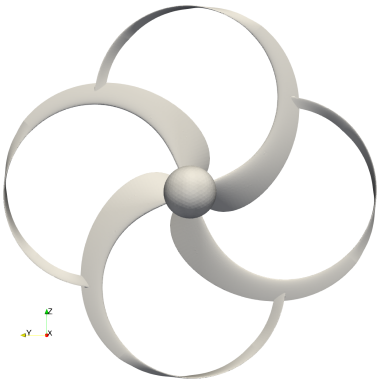
1. First Section: From the center of the rotor to $3\pi/4$ with twist angles calculated with BEM $\theta = \theta_{BEM}$;
2. Second Section: From $3\pi/4$ to π with twist angle calculated linearly from θ_{BEM} to 0;
3. Third Section: From π to $5\pi/4$ with twist angles calculated linearly from 0 to the inflow angle calculated by BEM $\pi - \phi_{BEM}$;



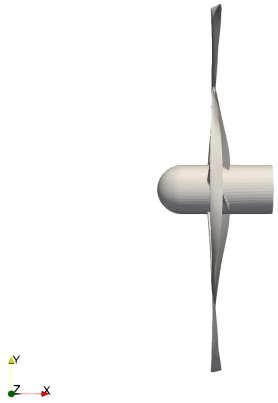
(a) Front view of Rotor II



(b) Side view of Rotor II



(c) Front view of Rotor III



(d) Side view of Rotor III

Figure 3.16 – Image of Rotor II and III

4. Fourth Section: From $5\pi/4$ to $3\pi/2$ with $\theta = \pi - \theta_{BEM}$, for the second geometry and $\theta = \pi - \phi_{BEM}$ for the third geometry.

The results of the two rotors may be seen in Figure (3.16).

4 Numerical Methodology

This chapter presents the numerical methodology used in this work in order to test the preliminary performance of the toroidal rotor geometry. The numerical simulations were conducted using the open-source CFD software OpenFOAM, using the Moving Reference Frame (MRF) approach in a steady-state simulation. There were in total 3 geometry variations tested, whose meshes were generated using CFMesh, for the first two and SnappyHexMesh for the third one. All geometries were tested using the K-ω SST turbulence model.

Rotor geometries I and II were tested using cases built to simulate the wind tunnel from the Energy and Environment Laboratory (*Laboratório de Energia e Ambiente* - LEA) from the University of Brasília (UnB), and were built by Matheus Nunes, a doctoral student from UnB. The last geometry was tested using a case built by the author. The latter will be discussed more deeply, since the main differences are the meshing method and the boundary condition for the walls, while LEA's case is a closed domain with a no-slip condition, the case built by the author is an open domain. The numerical schemes and solution algorithms are similar. The first case also has a cylindrical structure in the back to simulate a torque meter.

4.1 Introduction

The numerical methodology used in this work is based on OpenFOAM, an open-source computational fluid dynamics (CFD) toolbox. OpenFOAM is based on the finite volume method (FVM) and written in C++. The FVM is a discretization technique in which the domain is subdivided in small control volumes, commonly called cells. In each cell, volume properties are calculated through volume integrals resulting in a value of the property at the centroid, while the surface properties are calculated through the surface integral of the flux of each property through each of the faces, assigning thus a value to the property at each of the faces.

A general numerical simulation scheme can be summarized as:

1. Geometry generation;
2. Meshing;
3. Solving;
4. Post-process;

4.2 Geometry Description

There were 3 geometries tested in this work, the first one was a reverse-engineered geometry of the patented design of the toroidal rotor (SEBASTIAN; STREM, 2019), the second geometry was a corrected version of the first one so that the rotor could work as a wind turbine rotor. The third geometry was an attempt to improve the performance of the latter.

Property	Rotor I	Rotor II	Rotor III
Base axis of profile sections	x	z	z
Percentage of the blade using BEM	$\approx 90\%$	50%	75%
Minimal chord after maximal	$\bar{c}$	$\bar{c}$	$\bar{c}/2$
Diameter [m]	0.22	2.2	0.22
Scale[1]	1:1	1:1	1:10

Table 4.3 – Geometric Differences between rotors

4.3 Mesh

In this section, the meshes used in the tests will be described.

When it comes to generating the mesh for external flows, as it is the case of the current work, OpenFOAM has a couple of mesh generating solutions, the ones used for this work were `BlockMesh`, `SurfaceFeaturesExtract`, `SnappyHexMesh`, which work, respectively in the following manner: generating the background mesh, extracting the surface features of the geometry, subdividing the background mesh until the desired refinement level and trimming the mesh inside the geometry, moving vertices of the tetrahedra to the surface of the geometry, and then adding layers from the surface out in order to create a more refined mesh in the walls of the desired Geometry.

After generating the mesh, it is important to check its quality, since poor quality meshes are known to give results which are not representative of the physical problem, due to induced errors due to the geometry of the cell which exert influence on the numerical solutions, or even the low discretization of the mesh, which causes it to not capture relevant information about the flow. The latter is harder to evaluate, since the results and the quality of the mesh are essentially correct, the first problem on the other hand is measurable and needs to be addressed in order to facilitate the convergence of the results. For that reason, OpenFOAM is allows for the setting of mesh quality, which were presented in Table 4.4.

4.3.1 Mesh I

The first mesh has four regions of refinement and can be visualized in Figures 4.17 and 4.18.

Property	Parameter	Value
Maximal non-orthogonality	maxNonOrtho	65
Maximal face skewness at the boundaries	maxBoundarySkewness	20
Maximal face skewness in the domain	maxInternalSkewness	4
Maximal concavity of cells	maxConcave	80
Minimal Volume	minVol	disabled
Minimal cell twist angle	minTwist	0.01
Minimal determinant	minDeterminant	0.001
Minimal volume ratio	minVolRatio	0.01
Number of smooth sales	nSmoothScale	4
Error reduction	error reduction	0.75

Table 4.4 – Mesh Quality Parameters

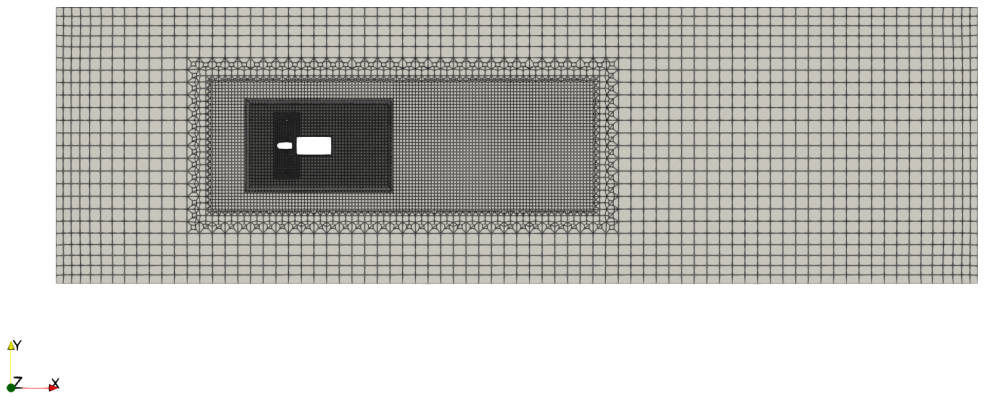


Figure 4.17 – Crossection of Mesh I

4.3.2 Mesh II Properties

The second mesh has three regions of refinement and can be visualized in Figures 4.19 and 4.20.

4.3.3 Mesh III

The mesh for the third geometry was generated using BlockMesh and SnappyHexMesh. It consists of a background consisting of a cuboid with a height of $2D$ and a width of $2D$ with a length of $3.0m$, where D is the diameter of the rotor. The mesh was later refined using SnappyHexMesh, which subdivides the background mesh until the desired refinement level is reached. In order to determine the rotating domain of the MRF, a cylinder was created with a radius of $0.145m$ and length of $0.145m$ with its center at the center of the

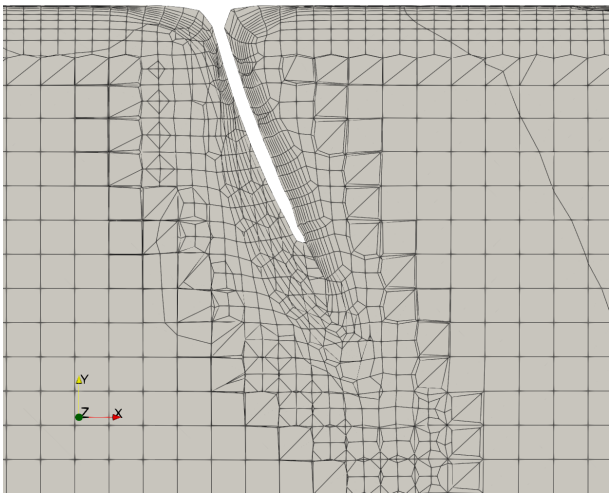


Figure 4.18 – Close-up on the crosssection of Mesh I

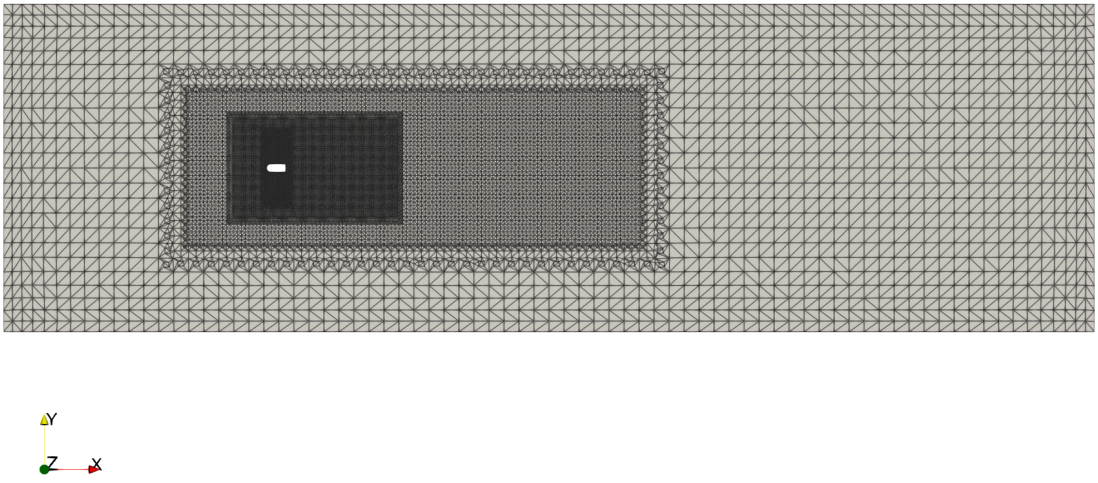


Figure 4.19 – Crossection of Mesh II

rotor. Aiming to capture the near wake of the rotor, the mesh was refined in the region of another cylinder with the front face at $-0.145m$ and the back face at $1.45m$ with a radius of $0.75D$. The resulting mesh can be observed in Figures 4.21 and 4.22.

The resulting mesh properties can be observed in Table 4.5. Although the mesh is not perfect, and the layering process was not successful, it can be said that the mesh

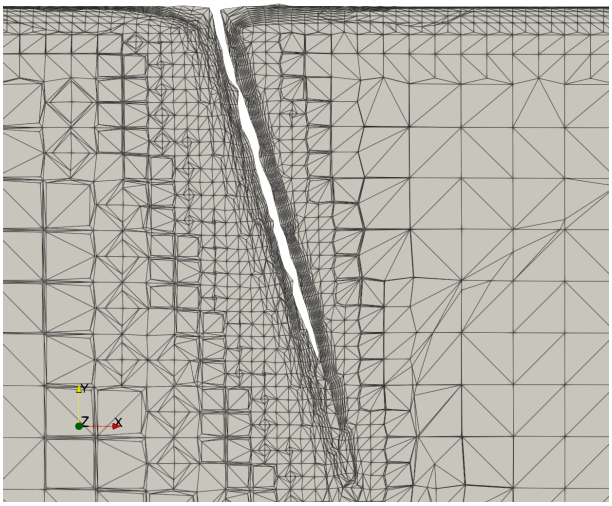


Figure 4.20 – Close-up on the cross-section of Mesh II

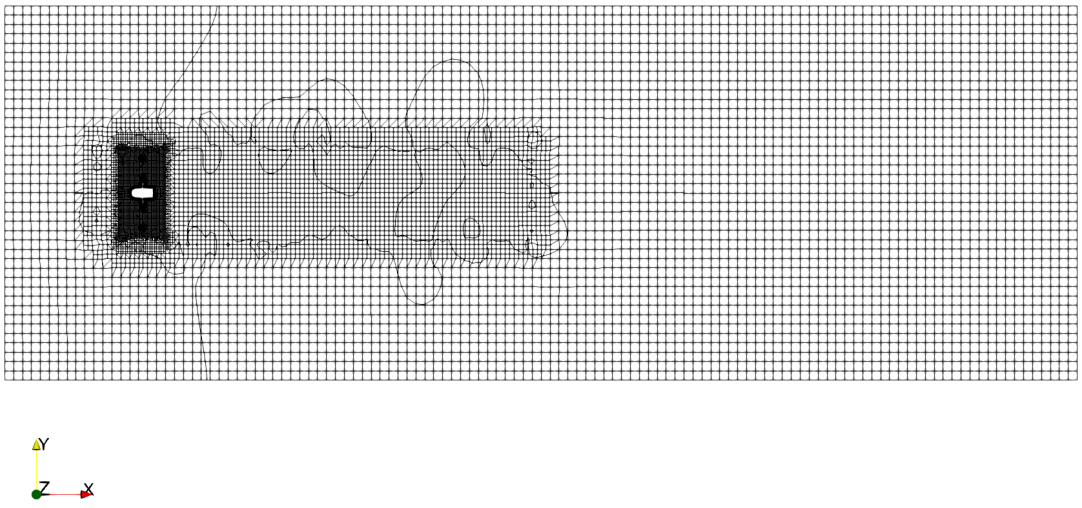


Figure 4.21 – Crossection of Mesh III

is nevertheless decent. And should be able to simulate the flow around the rotor with reasonable accuracy and convergence, keeping in mind the objective of this work, which is generating preliminary results for the developed geometry.

The boundaries of the fluid domain in OpenFOAM are defined as patches, which are named following the convention from the book of Versteeg and Malalasekera (2007). The

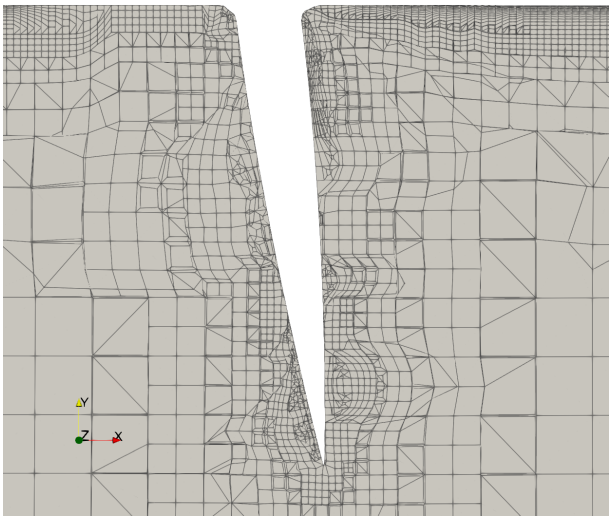


Figure 4.22 – Close-up on the cross-section of Mesh III

Property	Value
Number of cells	1.428.400
Number of faces	4.551.991
Number of nodes	1.713.785
Number of surface layers	20
First Layer thickness	1e-5m
Expansion ratio	1.2
Maximal non-orthogonality	65.6607
Number of highly non-orthogonal faces	0
Maximal face skewness	7.7778
Number of highly skewed faces	39

Table 4.5 – Mesh III Properties

patches are therefore north, south, inlet, outlet, top, and bottom, which are the boundaries with minimal z, maximal z, minimal x, maximal x, maximal y, and minimal y coordinates respectively. Additionally, the patch for the rotor is also defined. In this work, the north, south, top, and bottom patches will be called wall patches.

4.4 Boundary Conditions

An important part of the numerical simulation in regards to accuracy and convergence is the correct definition of the boundary conditions. In this section, the boundary conditions used in the simulations will be presented.

The boundary conditions for the simulation aim to model the flow around a free standing rotor in an open field, which means that the walls of the domain are subject to inflow and outflow of mass, and do not exert any influence in the the flow inside the fluid

domain, as is the case for confined rotors in wind tunnels.

For the velocity field, the inlet boundary condition was defined as a fixed value uniformly across the inlet patch, with a value of the free stream velocity $U_0 = 8m/s$ in the x axis.

The turbulent kinetic energy boundary condition is similar, an inlet condition, that is a fixed-value uniform field, however, this uniform field is also applied to the full fluid domain, defined as

$$k_0 = \frac{3}{2} I U_0^2, \tag{4.1}$$

where I is the turbulence intensity, which was set to $I = 0.5\%$, which is typical of wind tunnels. The boundary condition for the specific turbulent dissipation rate was defined in the same manner, with a uniform field however valued as

$$\omega_0 = \frac{k_0}{C_\mu^{0.25} c}, \tag{4.2}$$

where C_μ is the turbulent viscosity constant, which is usually set to $C_\mu = 0.09$, and c is the mean chord value.

The outlet boundary condition was defined as a constant pressure outlet, which models the relative atmospheric pressure, with a value of $p = 0Pa$. The wall patches were defined as symmetry planes, which means that the flow can cross the wall boundary freely. Finally, the rotor patch has a no-slip boundary condition, and of course uses also wall functions for the turbulence model, for both the k and ω. The boundary conditions used in the simulations are summarized in Table (4.6).

The fluid in the flow is assumed to be air at $25°C$ and $1atm$, which is considered incompressible at the conditions of the simulation. Additionally the third rotor is scaled down, thus in order to maintain the Reynolds number, and consequently the flow behavior, and the viscosity must be scaled down accordingly [2].

It is important to note that scaling down a rotor geometry for hydrokinetic turbines by a factor of 10 in order to perform tests in a wind tunnel (BRASIL JUNIOR et al., 2019), is not the same as scaling down a wind turbine, due to the fact that the kinematic viscosity of air and water differ by an order of magnitude, thus the Reynolds number of the flow around the rotor of a hydrokinetic turbine stays approximately the same, and the flow behavior is preserved, when cavitation phenomena are not present.

[2] If $L1 = 10L_2$ then

$$Re = \frac{L_1 u}{\nu_1} = \frac{L_2 u}{\nu_2} \Rightarrow \nu_2 = \frac{\nu_1}{10}$$

Patch	U	P	k	ω
Inlet	fixedValue	zeroGradient	fixedValue	zeroGradient
Outlet	inletOutlet	fixedValue	inletOutlet	zeroGradient
Wall	symmetryPlane	symmetryPlane	symmetryPlane	symmetryPlane
Rotor	MRFnoSlip	zeroGradient	kqRWallFunction	omegaWallFunction

Table 4.6 – Boundary Conditions for each Field

4.5 Numerical Schemes

The choice of an appropriate numerical scheme is of undeniable importance in the convergence and accuracy of the numerical simulations, since the numerical schemes are responsible for the discretization of the differential equations which models the physical problem. OpenFOAM allows for the choice of a vast number of numerical schemes each one of them with its advantages and disavantages (FOUNDATION, 2023).

The time discretization scheme used in this work was the steady-state solver, which sets the time derivative to zero.

In the finite volume method, values of the properties are calculated at the the centroid of the cell and the fluxes are calculated at the face of the cell, which means that the values of the properties must be interpolated to the faces of the cell. For the calculation of gradients the Gaussian linear interpolation and due to the poorer quality of the mesh, which is not fully orthogonal limiters were used. For the calculations of divergence terms, upwind schemes were used. The upwind discretization schemes are based on the direction of the information transport in the flow, as in the case of the simulation of a turbine rotor, the flow is mainly in the x direction, and therefore the flow transports information from the inlet to the outlet, the upwind schemes are the most appropriate. For the discretization of surface normal gradients and laplacian terms, limited corrected and limited corrected Gaussian linear schemes were employed with a limiter of 0.777. The full `fvSchemes` file can be referred to in Appendix B.1. These sets of schemes were chosen after a series of tests and based on the recommendations from WolfDynamic (2020), allowing for the convergence of the results.

4.6 Solver and Algorithm

The solvers used in this work were Stabilised preconditioned bi-conjugate gradient (PBiCGStab) for the pressure p equation, and the preconditioned bi-conjugate gradient (PBiCG) for the velocity U, turbulent kinetic energy k and specific turbulent dissipation rate ω equations.

In OpenFOAM version 11, the solver algorithm used for the solution is chosen relatively automatically based on the time discretization and the type of fluid flow being simulated. In this work, the steady state solver was used, with the Incompressible Navier-

Stokes equations, along with the MRF model for the rotation of the rotor, the K-ω SST turbulence model, and the PIMPLE algorithm for the pressure velocity coupling, which is a combination of the SIMPLE and PISO algorithms (VERSTEEG; MALALASEKERA, 2007). Since the flow is steady, the PIMPLE algorithm used in this work is essentially the SIMPLE algorithm.

The SIMPLE algorithm initially guesses the property fields, then solves the momentum equation for the velocity field, then the pressure correction equation, then corrects pressure and velocity fields, then finally solves all the transport equations iterating this process until the convergence criteria are met. The PISO algorithm works for the first part similar to the SIMPLE algorithm, after correcting the pressure and velocities however, it solves a second pressure correction equation and sets the pressure and velocity fields with the corrected values before solving all the transport equations (VERSTEEG; MALALASEKERA, 2007).

In OpenFOAM, the PIMPLE algorithm allows for the choice of the number of correctors, which were set to 1 outer correction consisting of the number of times which the PIMPLE algorithm is run before moving to the next time step; 2 correctors, which is the number of times the pressure correction equation is solved before calculating the final pressure field; and 1 non-orthogonal corrector, which is the number of times the non-orthogonal corrector is run before calculating the final pressure field. Therefore for each time step, the PIMPLE algorithm calculates the pressure four times, and is run once before moving to the next time step. It is important to remark that the momentum predictor is turned on.

The under-relaxation factor for all properties also ought to be set, they essentially limit the change in the value of the property from one iteration to another, these factors are highly influenced by the mesh quality and influence greatly the convergence of the results (FOUNDATION, 2023). The under-relaxation factors used in this work are 0.5 for the pressure field and 0.7 for the velocity U, turbulent kinetic energy k and specific turbulent dissipation rate fieldsω.

The complete set of parameters used in the `fvSolution` file can be refered to in Appendix B.2.

4.7 Numerical Procedure

The numerical procedure of preliminary tests performed on the rotor geometries were as follows:

- **Rotor I**: was tested once for 2.8 TSR;
- **Rotor II**: was tested once for 4.0 TSR;

- **Rotor III**: was tested for the 3.5, 4.0, 4.5, 6.0, 6.5, 8.0 TSR range.

The first two geometries were tested until 3.000 iterations, and the last was tested until convergence, monitored by the numerical residuals of iterations, and the mean square root (σ), namely the statistical error of the power coefficient C_P, as it is one of the parameters of interest and it needs to analyze the pressure and viscous tension distribution over the rotor surface.

5 Results

After running the numerical simulations as described in Chapter 4, the results were analyzed and the following conclusions were drawn.

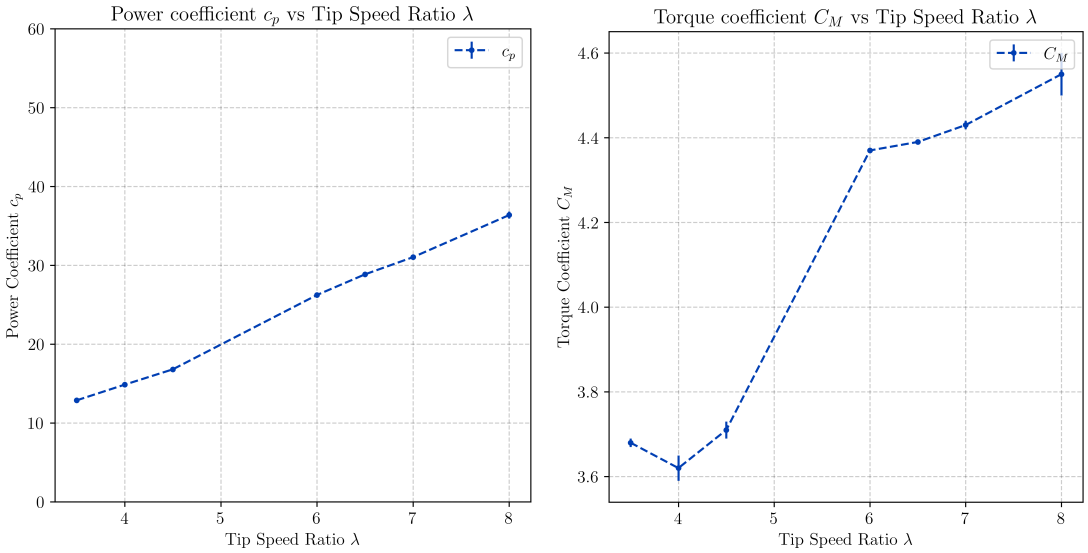
5.1 Torque and Power Coefficient Curves

The power coefficient C_P is an adimensional number that relates the power extracted from the fluid and the available power in the fluid, as already presented in eq. 2.5. In the context of wind and hydrokinetic turbine rotors, the C_P is used as a performance indicator, and the efficiency η of an axial rotor is measured as the ratio between the C_P and the maximal possible C_P, that is, the Betz limit. The power coefficient measured in the simulations 4.0 TSR is presented in Table 5.7.

	Rotor I	Rotor II	Rotor III
$C_P[\%]$	-17.51	6.00	14.87
$\sigma_{CP}[\%]$	–	–	0.13

Table 5.7 – Power coefficient for rotor III at 4.0 TSR.

The power coefficient curve for the rotor III is presented in Figure 5.23. It can be observed, differently from the power coefficient curve for straight blades as seen in Figure 3.8.



(a) Power Coefficient C_P (b) Torque Coefficient C_M

Figure 5.23 – Power and Torque coefficient curve for the rotor III

The torque coefficient curve indicates that the rotor is not extracting torque from the fluid efficiently, therefore, although the rotor is extracting energy from the fluid to rotate

itself, it is not converting this momentum into torque. Brasil Jr. (2019) have reported C_M values in the order of 40% for a hydrokinetic turbine rotor, which is a much higher value than the one presented here. One possible explanation for the low C_M is that the the curved of the blade, more specifically the top 25-50% of the blade represents momentum-wise additional pressure and viscous drag, therefore a large part of the force exerted by the fluid upon the blades is balanced by this additional drag component. This is further corroborated by the momentum data from rotor II, on which pressure momentum is of magnitude of $-4.8Nm$ and viscous momentum of order $2.4Nm$, which is a considerable ratio of almost 50%.

Preliminary tests with the third rotor geometry indicate that the ratio between pressure and viscous momentum is lower. At 3.5 TSR, the pressure momentum is of order $-6.0 \cdot 10^{-3}Nm$ and the viscous momentum is of order $3.8 \cdot 10^{-5}Nm$, which is only about 0.6% of the pressure momentum. This is further corroborated by the pressure distribution in the blade, as can be seen in Figure 5.29. This however may be due to the large y^+ values in the blade surface, which may be causing the solver to underestimate the viscous effects acting on the blade.

There are innumerable reasons why the power curve for the rotor could be behaving linearly, one explanation for it is that the rotor geometry works mainly on drag forces (from pressure) instead of lift forces, which is common for simpler rotor geometries. Another possible explanation is that the activation energy for the rotor is lower then the inlet energy, which would mean that the rotor hasn't achieved its peak efficiency, and that its rotational speed is still lower, which is unlikely.

5.2 Rotor Analysis

Due to the fact that the geometry tested in this work is non-conventional, it is worthwhile to analyse properties such as pressure p, turbulent kinetic energy k, turbulent viscosity ν_T, and the y^+ parameter, in order to investigate if the rotor is working according to expectations.

5.2.1 Rotor I

The first geometry tested was the rotor I, described in Section 3.6. I can be seen in Figure 5.24 that the pressure distribution is the opposite of what is expected from a turbine rotor, that is the pressure in the back is higher than the pressure in the front, which could indicate that the rotor is working as a compressor, rather than a turbine. This is further corroborated by the power coefficient, which is negative, indicating the necessity of adding energy to the system.

Therefore, the geometry was modified in order to orient the profiles in the proper manner. The new geometry is presented in Section 3.7.

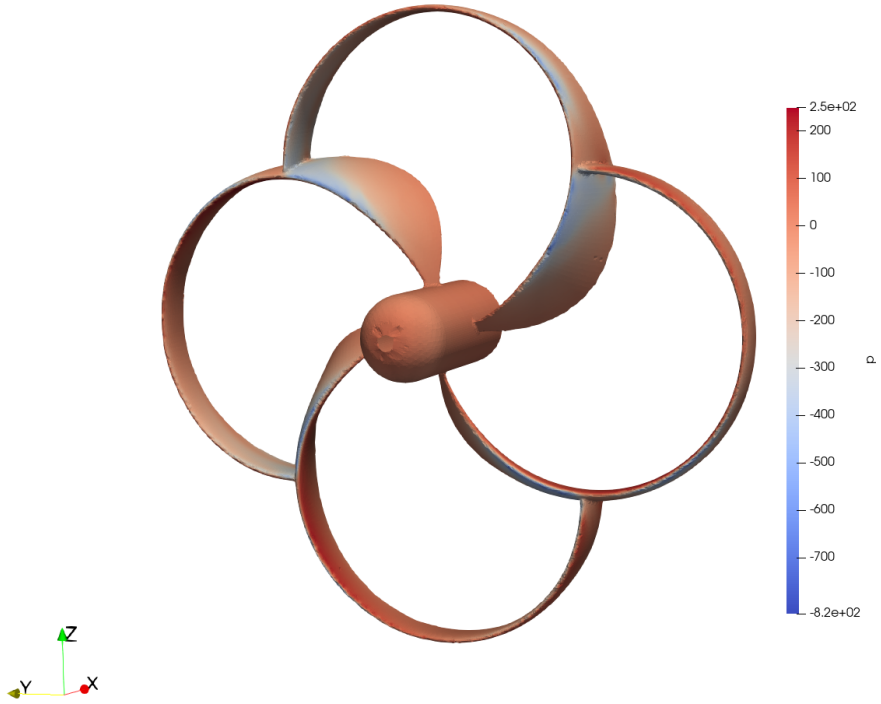
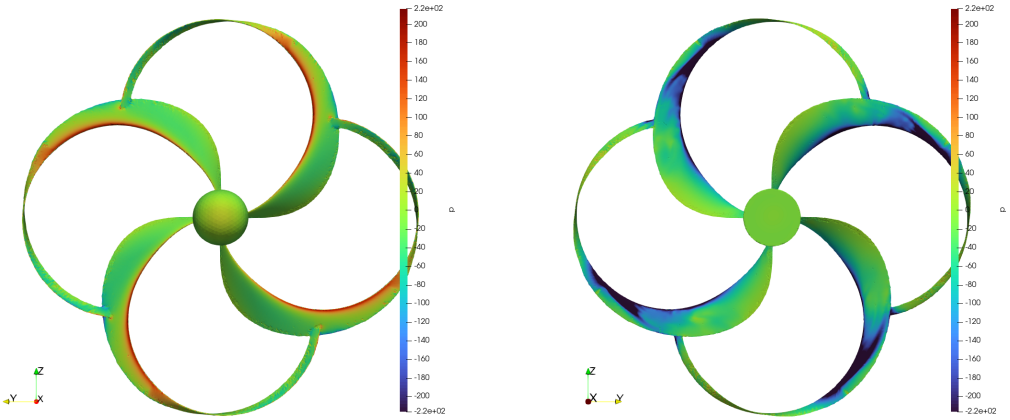


Figure 5.24 – Rotor I Pressure Distribution

5.2.2 Rotor II

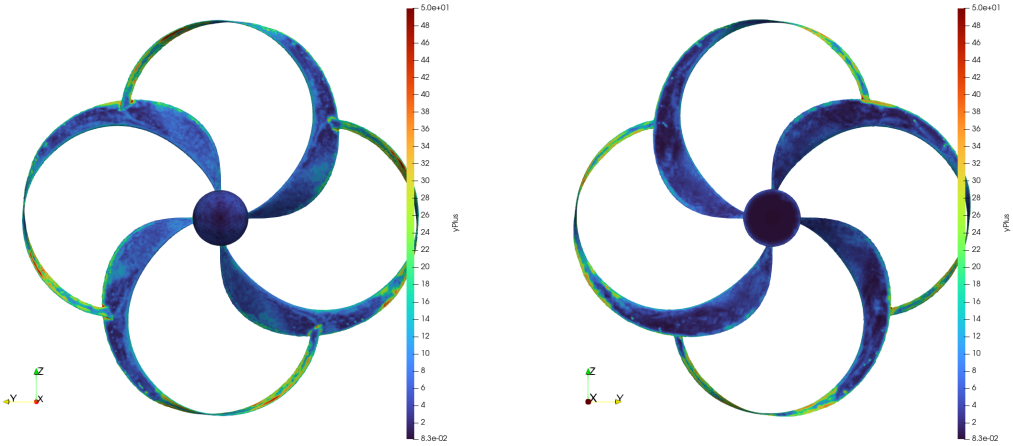
Figure 5.25 presents the pressure distribution around rotor II, where it can be observed a high-pressure region at the pressure side leading edge, and a low-pressure region at the suction side. It can also be seen that in the connection of the blades both sides present a relatively high pressure, which indicates pressure drag. In the back of the rotor, a large low-pressure region can be observed before the linear transition region, which may indicate a stall condition.

Figure 5.26 presents the y^+ distribution around the rotor, where it can be observed that the y^+ is higher in the second and third section of the blade. High values of y^+ represent regions with low accuracy in the numerical solution of the near wall region. The highest value of y^+ in the rotor surface is of order $y^+ \propto 160$, which is a compromise between accuracy and computational cost, justified by the fact that these numerical simulations are preliminary and the main objective is to investigate the overall behavior of the rotor, rather than having high accuracy in the results.



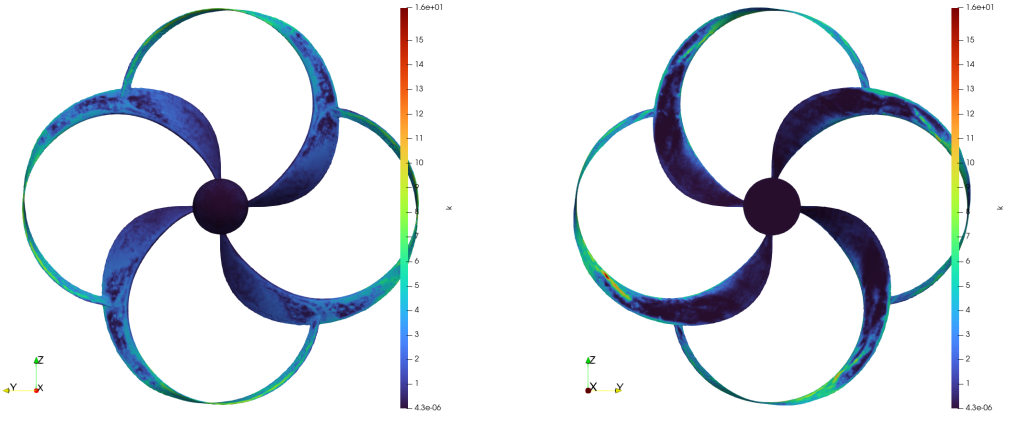
(a) Front View (b) Back View

Figure 5.25 – Rotor II at 4.0 TSR Pressure p ranging from -220 to $220Pa$



(a) Front View (b) Back View

Figure 5.26 – Rotor II at 4.0 TSR y^+ ranging from 0 to 50



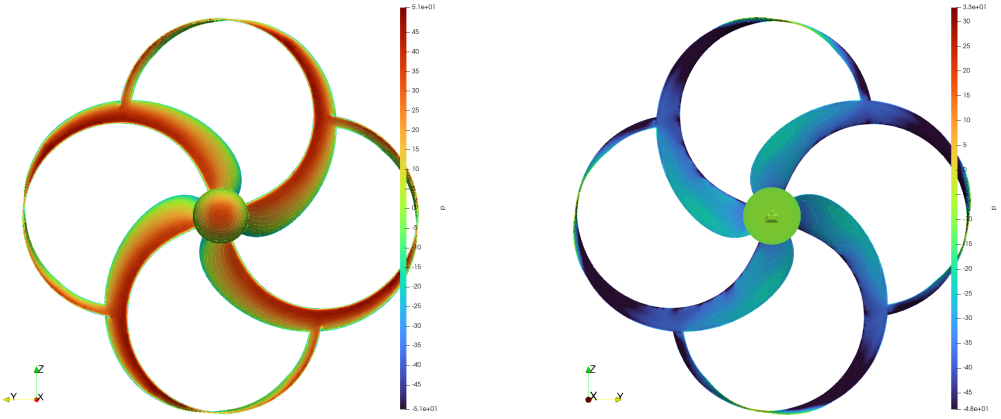
(a) Front View (b) Back View

Figure 5.27 – Rotor II at 4.0 TSR turbulent kinetic energy k ranging from $4.3 \cdot 10^{-6}$ to 16

5.2.3 Rotor III

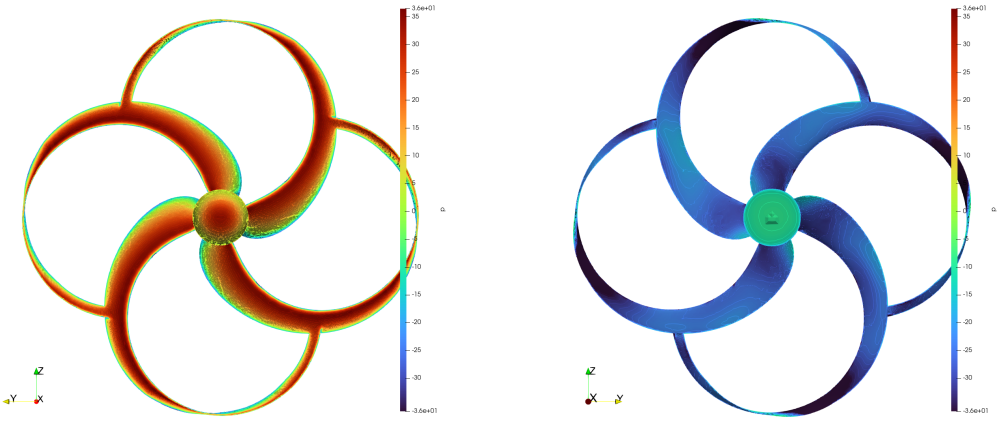
Observing the pressure distribution around rotor III, it is possible to observe that the pressure is higher in the front of the rotor, and lower in the back, which is expected for a turbine rotor, as the extraction of energy using torque will generate a pressure drop in the back of the rotor. It can also be observed that the leading edge of the rotor has a higher pressure than the trailing edge, which is also desirable.

Additionally from the pressure it is possible to observe that there is a lower pressure region at the top of the curve of the blade, where the twist angles of the profiles are higher, than what is calculated with BEM, due to the linear transition between sections of the blade, which is expected because it induces a stall condition. It can also be observed that in the last section of the blade close to the intersection, the pressure gradient between the suction and pressure sides of the blade are maintained. The pressure distribution is presented in Figures 5.28 and 5.29.



(a) Front View (b) Back View

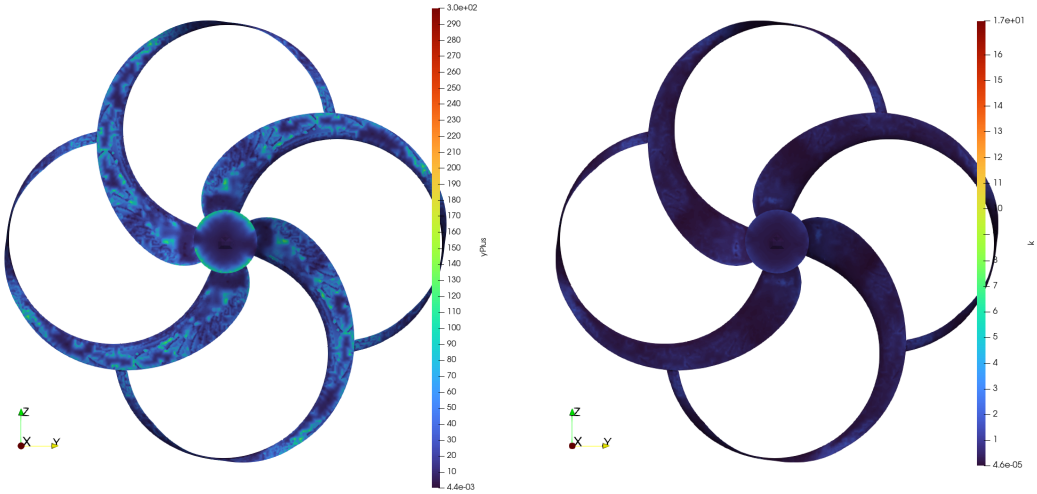
Figure 5.28 – Rotor III at 8.0 TSR Pressure p ranging from -51 to $51Pa$



(a) Front View (b) Back View

Figure 5.29 – Rotor III at 3.5 TSR Pressure p ranging from -36 to $36Pa$

The turbulent kinetic energy k suggests that turbulence is not produced in the surface of the rotor at 3.5 TSR, indicating that there was little separation of flow in the blade surface, which is a good indication that the rotor is working as expected, as it can be seen in Figure 5.30b. The y^+ distribution in the rotor blades go up to 420, which is a high value, for reference, good meshes for solving turbulent flows with $k - \omega$ SST should have y^+ values between 30 and 100.



(a) y^+ at the back

(b) Turbulent kinetic energy k at the back

Figure 5.30 – Rotor III at 3.5 TSR y^+ and k

5.3 Wake Visualization

Figure 5.31 presents the pressure distribuition in the wake of the rotor I, and it can be observed that the pressure in the wake is reversed, due to the operation as a propeller, rather than a turbine. It can also be observed the influence of the nacelle in the wake of the rotor.

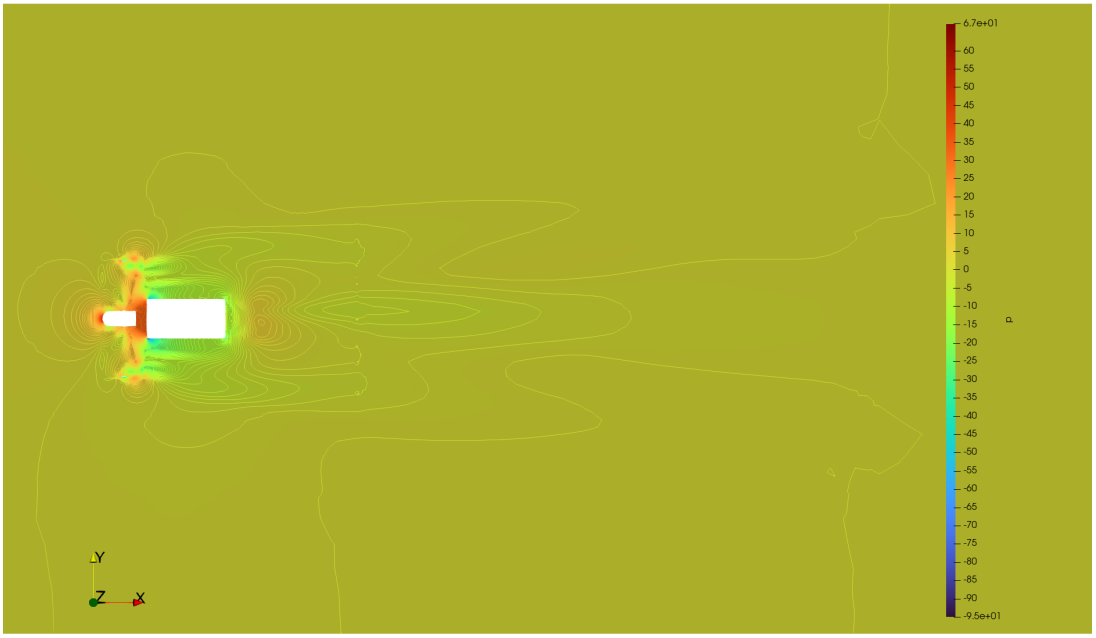


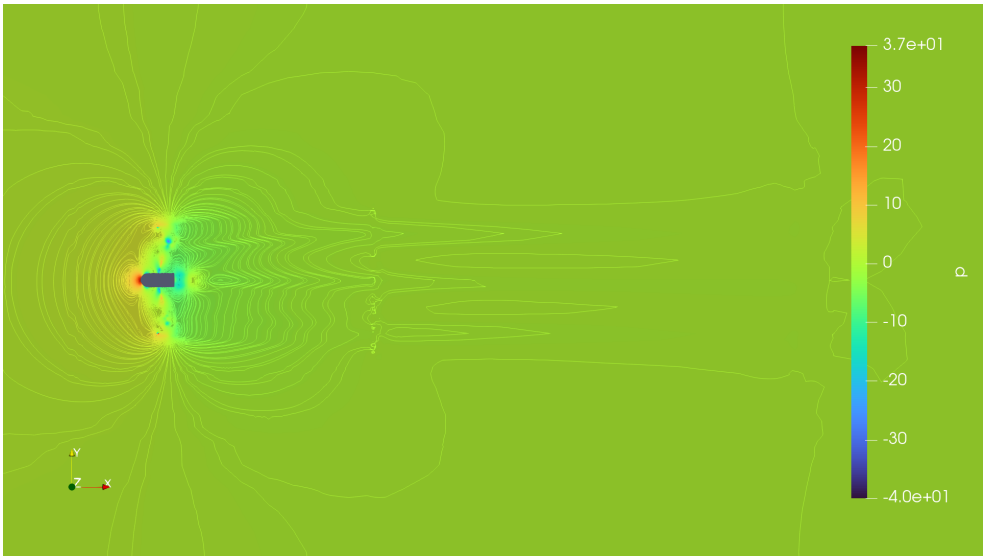
Figure 5.31 – Pressure Distribution and Contours for Rotor I at 2.8 TSR

The wake of the rotor is presented in Figure 5.32b, and it displays typical behavior of a turbine rotor, with a velocity deficit being present in the wake, and the wake being wider than the rotor itself, due to the induced divergence of the flow upstream by the presence of the rotor. It is also possible to observe the presence of the "center blade vortices", in the pressure fluctuations in the wake.

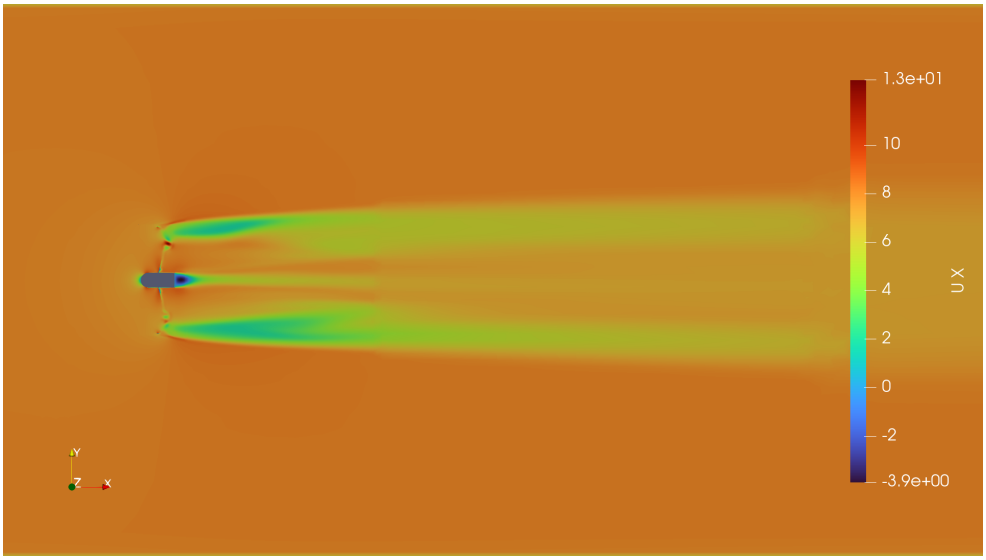
Figures 5.33a and 5.33b present the turbulent kinetic energy k field and the turbulent viscosity ν_T in the wake of rotor II at 4.0 TSR, respectively, and it can be observed that the wake is highly turbulent.

Figures 5.34a and 5.34b present the pressure distribution and the velocity field in the wake of rotor III at 6.5 and 3.5 TSR, respectively.

Analysing cross-sections of the wake, however, and plotting a velocity profile in the wake, it is possible to observe, in Figure 5.35, that there are three peaks of velocity deficit, which is also expected in straight blade geometries, in this case, however, the peaks are not occurrent in the wing tip, due to wing tip vortices formation, rather they appear in about the middle of the blade for the fluid flows from the pressure side of one blade to the suction side



(a) Pressure Distribution p and Contours

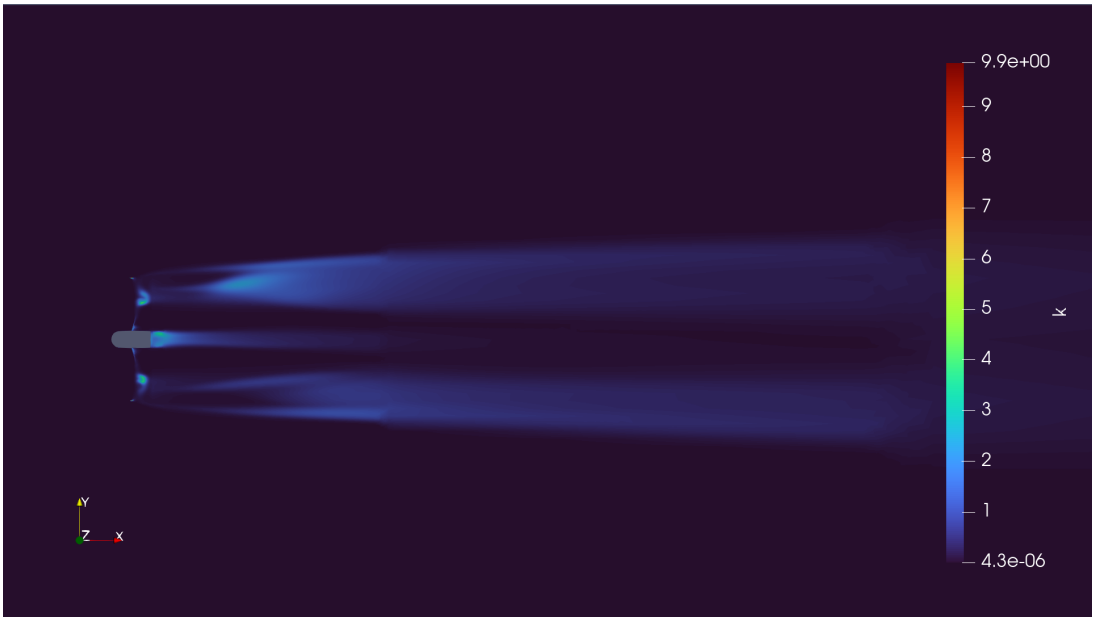


(b) Velocity Field U_x

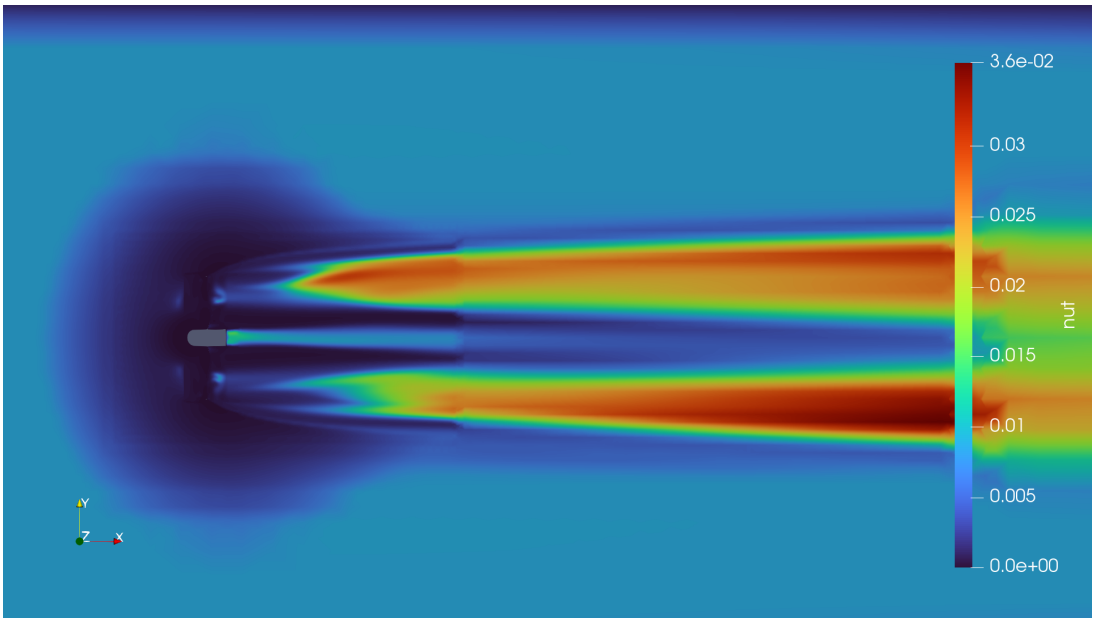
Figure 5.32 – Wake Visualization for Rotor II at 4.0 TSR

of the adjacent. The position of the center and intensity of the vortex must be a function of the pressure distribution of both blades, and should be further investigated.

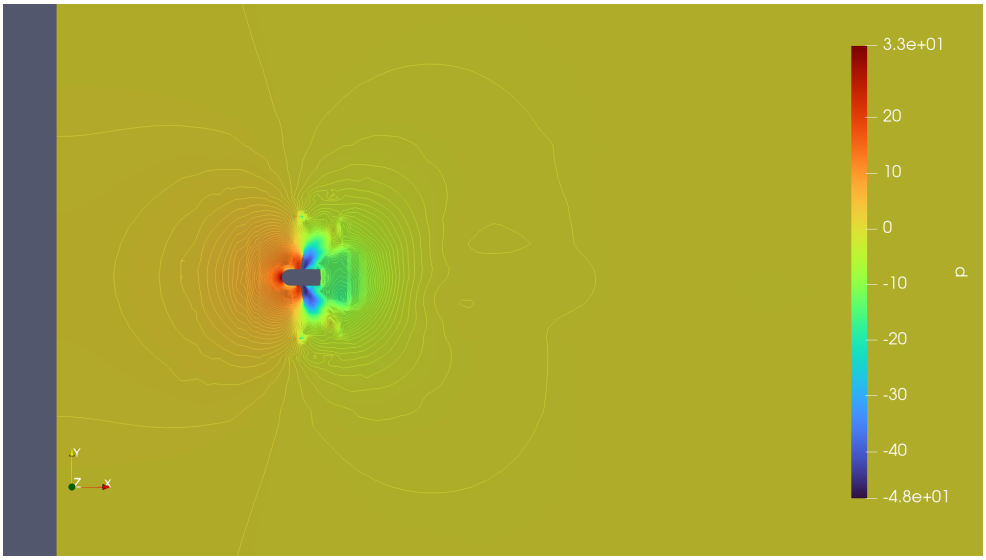
Figure 5.36a shows the contours for the induction factor a of rotor III at the rotor plane, and indicates how the velocity U_x is reduced in the rotor plane, and gives a rough idea of the velocity deficit or surplus in different parts of the rotor plane. Near the rotor walls, lines are tighter together, due to the reduction in velocity. Additionally, Figure 5.36b presents the induction factor contours for a plane at $x = -0.45R$ (upstream), which can be used as an indication of the validity of Momentum Theory, since it is not valid for induction factors greater than 0.5 (HANSEN, 2015).



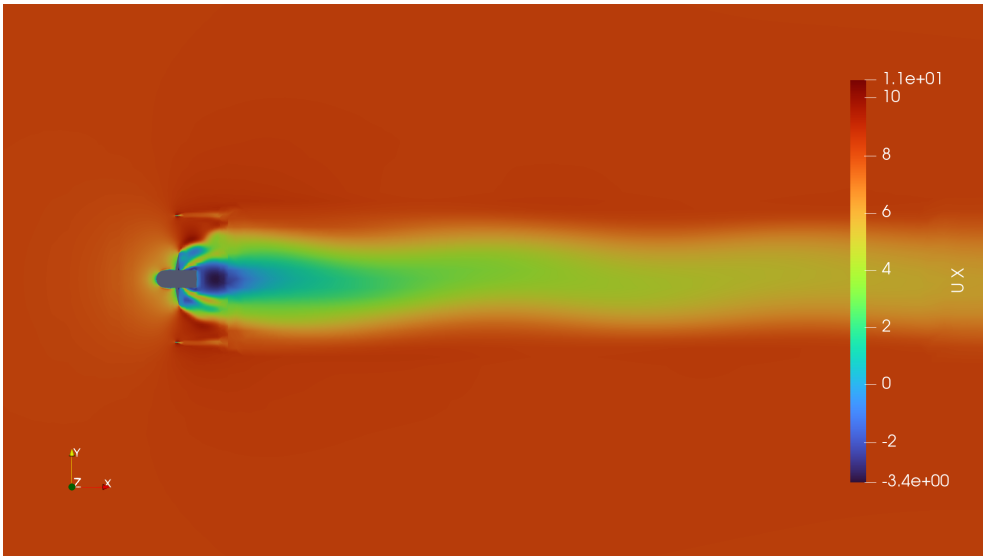
(a) Turbulent Kinetic Energy k for Rotor II at 4.0 TSR



(b) Turbulent Viscosity ν_T for Rotor II at 4.0 TSR



(a) Pressure Distribution p and Contours at 6.5 TSR



(b) Velocity Field U_x at 3.0 TSR

Figure 5.34 – Wake Visualization for Rotor III at 3.5 TSR

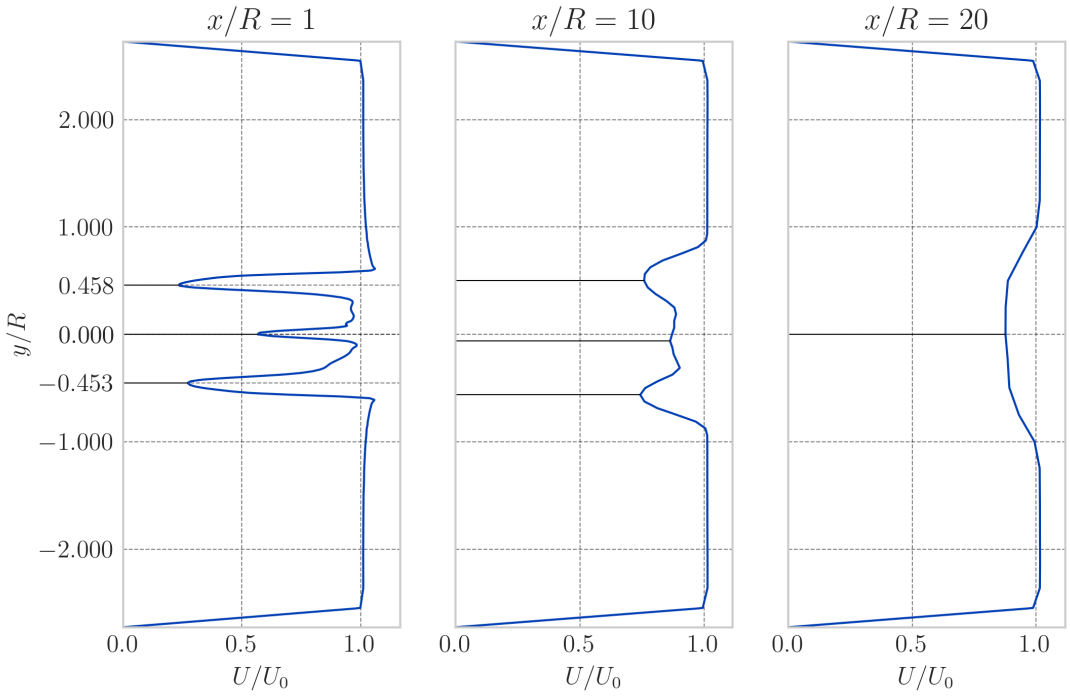
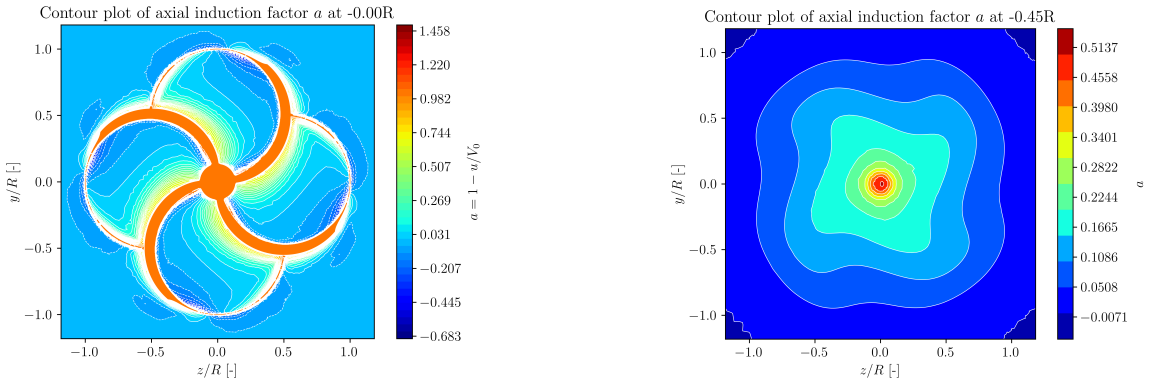


Figure 5.35 – Velocity Profiles U_x at $x = 1R$, $10R$, $20R$ for the wake in Rotor II



(a) at the rotor plane $x = 0$

(b) at $x = -0.45R$

Figure 5.36 – Induction Factor a Contours for Rotor III at 3.5 TSR

6 Conclusions

This work used BEM and CFD to develop, simulate and analyze a toroidal rotor geometry for a wind turbine. The geometry was developed using a NACA 0015 airfoil and a quad-toroidal circular geometry using Python. The numerical simulation was performed using OpenFOAM, and the results were analyzed using ParaView and Python.

The torque coefficient curve indicates that the rotor is not extracting torque efficiently. The power coefficient curve indicates that the rotor indeed acts as a turbine, extracting energy from the fluid.

The pressure distribution around the rotor indicates that the rotor is working as expected, and the changes in geometry did actually improve the performance of the rotor. However the efficiency of the rotor is still low, and further research is necessary before a final conclusion regarding the viability to be drawn.

The wake of the rotor indicates that the wing tip vortices in toroidal geometries are formed due to the difference in pressure between one blade's pressure side and the other blade's suction side. The wake also indicates high complexity, with the presence of secondary vortices and a high level of turbulence. The complexity of the wake may challenge the usage of such rotors in wind farms, since the wake may interfere with the performance of others.

The findings of this work indicate that the toroidal rotor is a viable concept for wind turbines, but further research is necessary to improve the performance of the rotor and to understand the wake of the rotor.

7 Next Studies

The following are suggestions for future studies regarding the toroidal rotor:

- **Viability of BEM for 3D geometries:** BEM is a powerful tool for designing wind turbines, however, its applicability to 3D geometries is still under debate. Is the calculation of the induction factor in 2D geometries also valid for 3D geometries? How does the 3D geometry affect the induction? factor? How do the induction factors affect the induced angle on profiles? which are not perpendicular to the radius? How does the 3D geometry affect the torque and power coefficients?
- **Geometric optimization:** The geometry of the toroidal rotor is based on the geometry of the MIT toroidal propeller, therefore it is not yet known if there is a better geometry for the toroidal rotor. The suggestion would be to try to curve the blades downstream.
- **Wake analysis:** The wake of the toroidal rotor is very complex, therefore analyzing the wake thoroughly and evaluating the impact of the wake on other rotors is a necessary step before the toroidal rotor can be used in wind farms.

References

ANDERSON, J. **Fundamentals of Aerodynamics**. McGraw Hill, 2011. Cit. on p. 23.

BETZ, A. Einführung in die Theorie der Flugzeug-Tragflügel. **Naturwissenschaften**, Springer-Verlag Berlin/Heidelberg, v. 6, n. 38, p. 557–562, 1918. Cit. on p. 18.

BETZ, A. **The theory of the screw propeller**. 1922. Cit. on p. 22.

BRASIL JUNIOR, A. C.; MENDES, R. C.; WIRRIG, T.; NOGUERA, R.; OLIVEIRA, T. F. On the design of propeller hydrokinetic turbines: the effect of the number of blades. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, Springer, v. 41, p. 1–14, 2019. Cit. on pp. 25, 26, 32, 38, 40–42, 54, 59.

BREAKTHROUGH ENERGY. Available at: https://breakthroughenergy.org/our-approach/grand-challenges/ – accessed on 27 Jul. 2023. 2023. Cit. on p. 14.

FOUNDATION, T. O. **OpenFOAM.org User Guide**. July 2023. Cit. on pp. 55, 56.

FROUDE, W. **On the elementary relation between pitch, slip, and propulsive efficiency**. 1920. Cit. on p. 15.

GIPE, P.; MÖLLERSTRÖM, E. An overview of the history of wind turbine development: Part I—The early wind turbines until the 1960s. **Wind Engineering**, SAGE Publications Sage UK: London, England, v. 46, n. 6, p. 1973–2004, 2022. Cit. on p. 14.

GLAUERT, H. Airplane propellers. **Aerodynamic theory**, Julius Springer, 1935. Cit. on p. 15.

HANSEN, M. O. **Aerodynamics of wind turbines**. Routledge, 2015. Cit. on pp. 21, 22, 24, 25, 66.

HAU, E. **Windkraftanlagen: Grundlagen, Technik, Einsatz und Wirtschaftlichkeit**. Springer-Verlag, 2017. Cit. on p. 17.

JOHNSON, D.; GU, M.; GAUNT, B. Wind Turbine Performance in Controlled Conditions: BEM Modeling and Comparison with Experimental Results. **International Journal of Rotating Machinery**, v. 2016, p. 1–11, Jan. 2016. DOI: 10.1155/2016/5460823. Cit. on p. 26.

KOLMOGOROV, A. N. The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers. **Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences**, The Royal Society London, v. 434, n. 1890, p. 9–13, 1991. Cit. on p. 28.

LINCOLN LABORATORY - MIT. **Innovation Highlight**: Toroidal Propeller. 2022. Available at: https://www.ll.mit.edu/sites/default/files/other/doc/2023-02/TVO_Technology_Highlight_41_Toroidal_Propeller.pdf – accessed on 17 Jul. 2021. Cit. on p. 19.

MAALAWI, K. Y.; BADAWY, M. T. A direct method for evaluating performance of horizontal axis wind turbines. **Renewable and Sustainable Energy Reviews**, Elsevier, v. 5, n. 2, p. 175–190, 2001. Cit. on p. 25.

MEHDIPOUR, R. **Simulating propeller and propeller-hull interaction in openFOAM**. 2014. Cit. on p. 31.

MENDES, R. C. F.; MACIAS, M. M.; OLIVEIRA, T. F.; BRASIL ANTONIO C. P., J. A Computational Fluid Dynamics Investigation on the Axial Induction Factor of a Small Horizontal Axis Wind Turbine. **Journal of Energy Resources Technology**, v. 143, n. 4, p. 041301, Aug. 2020. https://doi.org/10.1115/1.4048081 – accessed on 2nd Dec. 2023. Cit. on p. 42.

MENTER, F. R. Two-equation eddy-viscosity turbulence models for engineering applications. **AIAA journal**, v. 32, n. 8, p. 1598–1605, 1994. Cit. on p. 30.

RODRIGUEZ, S.; JAWORSKI, J.; MICHOPOULOS, J. Stability of helical vortex structures shed from flexible rotors. **Journal of Fluids and Structures**, v. 104, p. 103279, July 2021. DOI: 10.1016/j.jfluidstructs.2021.103279. Cit. on pp. 16, 17.

SEBASTIAN, T.; STREM, C. **United States Patent Application Publication**. May 2019. US2019/0135410 A1. Cit. on pp. 18, 20, 32, 49.

SHARROW MARINE: Performance Reports - 36' Twin Vee with Twin Yamaha 300 HP. Available at: https://sharrowmarine.com/performance-reports/2022/12/30/36-twin-vee-with-twin-yamaha-300-hp – accessed on 14 Jul. 2021. 2022. Cit. on pp. 18, 19.

SORENSEN, J. N.; SHEN, W. Z. Numerical modeling of wind turbine wakes. **J. Fluids Eng.**, v. 124, n. 2, p. 393–399, 2002. Cit. on p. 30.

VERMEER, L.; SØRENSEN, J. N.; CRESPO, A. Wind turbine wake aerodynamics. **Progress in aerospace sciences**, Elsevier, v. 39, n. 6-7, p. 467–510, 2003. Cit. on pp. 15, 27.

VERSTEEG, H.; MALALASEKERA, W. **An Introduction to Computational Fluid Dynamics: The Finite Volume Method**. Pearson Education Limited, 2007. ISBN 9780131274983. Available from: <https://books.google.com.br/books?id=RvBZ-UMpGzIC>. Cit. on pp. 27, 52, 56.

WOLFDYNAMICS. **OpenFOAM training material**. 2020. Cit. on p. 55.

Appendix

Appendix A – Code

A.1 Profile Class

Code A.1 – Profile Class

```
import numpy as np
from source.tools import geo_oper as geo
from matplotlib import pyplot as plt


class Profile:
    def __init__(self, profile):

        self.chord = 1 # >> [m] ----------------------- chord
           length
        self.AoA = 0 # >> [rad] ----------------------- attack
           angle
        self.AoT = 0 # >> [rad] ----------------------- twist angle
        self.AoC = 0 # >> [rad] ----------------------- camber
           angle
        self.origin = np.array([0.,0.,0.]) # >> [m] --- origin of
           the profile
        # setting the profile coordinates
        self.profile = self._get_profile(profile)
        # translating profile to center the aerodynamic center
        self.profile[:,0] -= 0.25
        self.set_attack_angle(-np.pi/2)

    def _get_profile(self, profile):
        if type(profile) is str:
            coord_2d = np.loadtxt(profile, unpack=True)
        else:
            coord_2d = profile
        coord_3d = np.zeros((len(coord_2d[0]), 3))
        coord_3d[:,0] = coord_2d[0]
        coord_3d[:,1] = coord_2d[1]
        return coord_3d

    def set_attack_angle(self, AoA) -> None:
        self.AoA = AoA
        self.profile -= self.origin
        self.profile = geo.rotate(self.profile, self.AoA, 2)
        self.profile += self.origin

    def set_twist_angle(self, AoT) -> None:
        self.AoT = AoT
        self.profile -= self.origin
```

```
39          self.profile = geo.rotate(self.profile, self.AoT, 1)
40          self.profile += self.origin
41
42      def set_camber_angle(self, AoC) -> None:
43          self.AoC = AoC
44          self.profile -= self.origin
45          self.profile = geo.rotate(self.profile, self.AoC, 0)
46          self.profile += self.origin
47
48      def set_chord(self, chord) -> None:
49          self.chord = chord
50          self.profile -= self.origin
51          self.profile *= self.chord
52          self.profile += self.origin
53
54      def translate(self, origin) -> None:
55          self.origin[:] += np.array(origin)[:]
56          self.profile += self.origin
57
58      def set_profile(self, props, degrees=False) -> None:
59          if degrees:
60              props["AoA"] = np.radians(props["AoA"])
61              props["AoT"] = np.radians(props["AoT"])
62              props["AoC"] = np.radians(props["AoC"])
63
64          self.set_attack_angle(props["AoA"])
65          self.set_twist_angle(props["AoT"])
66          self.set_camber_angle(props["AoC"])
67          self.set_chord(props["chord"])
68          self.translate(props["origin"])
69
70      def scale(self, scale) -> None:
71          try:
72              len(scale)
73          except TypeError:
74              scale = np.array([scale, scale, scale])
75          self.profile -= self.origin
76
77          self.profile[:,0] *= scale[0]
78          self.profile[:,1] *= scale[1]
79          self.profile[:,2] *= scale[2]
80
81          self.profile += self.origin
82
83  if __name__ == "__main__":
84      naca_prof = np.loadtxt("./profiles/naca0015.dat", unpack=True)
85
86      airfoil = Profile(naca_prof)
87      props = {"AoA":5, "AoT":0, "AoC":0, "chord":1,
            "origin":[0,0,0]}
88      airfoil.set_profile(props, degrees=True)
89
```

```
    plt.plot(airfoil.profile[:,0], airfoil.profile[:,1])
    plt.axis('equal')
    plt.savefig("./test.png")
```

A.2 XFOIL Automation Module

Code A.2 – XFOIL Automatino

```
import os
import time
import subprocess

def init() -> None:
    """_summary_ Initialize the Xfoil environment"""
    # change this for the propper command in your system
    # xQuartz is the display in the environment
    os.system('open -a XQuartz')
    # this is the display port
    os.environ["DISPLAY"] = '127.0.0.1:0.0'


def clean_all(profile:str, Re:float) -> None:
    """_summary_ Clean all files generated by Xfoil

    Args:
        profile (str): NACA profile code
        Re (float): Reynolds number
    """
    if not os.path.exists("./xfoil"): os.mkdir("./xfoil")
    else:
        os.system(f"rm -r ./xfoil/dump.txt")
        os.system(f"rm -r ./xfoil/input.sh")
        os.system(f"rm -r ./xfoil/{profile}_{Re:.0f}.txt")

def write_sh(profile:str, alpha_i:float, alpha_f:float,
   alpha_step:float,
             Re:float, n_iter=1000) -> None:
    """_summary_ Write the input.sh file for Xfoil

    Args:
        profile (str): NACA profile code
        alpha_i (float): initial angle of attack
        alpha_f (float): final angle of attack
        alpha_step (float): angle of attack step
        Re (float): Reynolds number
        n_iter (int, optional): Number of maximum iterations.
           Defaults to 0.

    Returns:
        None
```

```
    """

    clean_all(profile, Re)

    ### Change absolute path to the current directory ###

    path = "/Users/felipeandrade/github/UnBEM"
    profile_path = "profiles/intermediary_profiles"

    with open(f"./xfoil/input.sh", 'w') as f:
        try:
            int(profile)
            f.write(f"naca {profile}")
        except TypeError:
            f.write(f"load ./{profile_path}/{profile}.dat\n")
            f.write(f"{profile}\n")
        f.write('PPAR\n')
        f.write("N 360\n\n\n")
        f.write("PANE\n")
        f.write('OPER\n')
        f.write(f'Visc {Re}\n')
        f.write(f'PACC\n')
        f.write(f'{path}/xfoil/{profile}_{Re:.2e}.txt\n')
        f.write(f'{path}/xfoil/dump.txt\n')
        if n_iter != 0: f.write(f'ITER {n_iter}\n')
        f.write(f'ASeq {alpha_i} {alpha_f} {alpha_step}\n')
        f.write('\n\n')
        f.write('quit\n')


def xfoil(profile: str, alpha_i: float, alpha_f: float,
    alpha_step: float,
           Re: float, n_iter: int = 0) -> None:
    """
    Runs Xfoil with the given parameters.

    Args:
        profile (str): The name of the airfoil profile file to be
            used by Xfoil.
        alpha_i (float): The initial angle of attack in degrees.
        alpha_f (float): The final angle of attack in degrees.
        alpha_step (float): The step size between consecutive
            angle of attack
        values in degrees.
        Re (float): The Reynolds number.
        n_iter (int, optional): The number of iterations to be
            performed by
        Xfoil. Defaults to 0.

    Returns:
        None

```

```
    Example usage:
    >>> xfoil("0012", 0, 10, 1, 100000)
    """
    write_sh(profile, alpha_i, alpha_f, alpha_step, Re, n_iter)

    ### path to xfoil ###

    path_to_xfoil = "../../Xfoil/bin/xfoil"

    subprocess.call(f"{path_to_xfoil} < ./xfoil/input.sh",
        shell=True)
    time.sleep(2)

if __name__ == "__main__":
    xfoil("4415", 0, 20, .05, 5e5, 2000)
```

A.3 UnBEM Implementation

Code A.3 – UnBEM Implementation in Python

```
import numpy as np
import os

def algorithm_1(TSR:float, radii:np.ndarray, toroidal=False):
    """_summary_ Calculates the indcution angles of given rotor
        and flow
    conditions

    Args:
        omega (float): Angular velocity (rad/s)
        V_0 (float): Inflow velocity (m/s)
        R (float): Rotor radius (m)
        r_hub (float): Hub radius (m)
        N_sec (int): Number of sections (ad.)

    Returns:
        tuple: axial induction angle, tangentcial induction angle,
            inflow angle

    """
    # divides the rotor radius into sections in a straight line
    R = radii[-1]
    N = len(radii)
    # admensional radial rotor speed
    lambda_r = np.zeros(N)
    # lambda_r = lambdas * radii/R
    a = np.zeros(N)
    at = np.zeros(N)
    phi = np.zeros(N)

```

```
29      for i in range(N):
30          if radii[i] == 0.0: radii[i] = 1e-05
31          lambda_r = TSR * radii / R
32          Lambda = np.sqrt(1+(lambda_r[i]*lambda_r[i]))
33
34          theta_plus = (1/3) * np.arccos(1/Lambda)
35          theta_minus = (1/3) * np.arccos(-1/Lambda)
36
37          a[i] = 0.5 * (1 - Lambda * (np.cos(theta_plus) -
               np.cos(theta_minus)))
38          at[i] = (1-(3*a[i]))/((4*a[i])-1)
39
40          phi[i] = np.arctan((1-a[i])/((1+at[i])*lambda_r[i]))
41
42      return {"a":a, "at":at, "phi":phi}
43
44  def get_optimal(file:str):
45      """Get optimal Ao, CLo, CDo values from xfoil polar data
46
47      Args:
48          file (str): xfoil polar data file path
49
50      Returns:
51          dict: Ao: optimal alpha,
52              CL: optimal lift coefficient,
53              CD: optimal drag coefficient,
54      """
55
56      polar_data = np.loadtxt(file, skiprows=12)
57      alpha = polar_data[:,0]
58      cl = polar_data[:,1]
59      cd = polar_data[:,2]
60      max_index = np.argmax(cl/cd)
61
62
63      return {"Ao":np.radians(alpha[max_index]),
64              "CLo":cl[max_index],
65              "CDo":cd[max_index]}
66
67  def _tip_correction(Nb, r, R, phi, R_hub=0.0):
68      """ Calculates a correction for the tip according to prandtls
           law
69
70      Args:
71          Nb (int): number of blades
72          r (float): radius of the section (m)
73          R (float): radius of the rotor (m)
74          phi (float): inflow angle (rad/s)
75
76      Returns:
77          float: tip correction coefficient
78      """
```

```
79      ff = 0.5*Nb*(1-(r/R))
80      ff /= (r/R)*np.sin(phi)
81      F_tip = (2/np.pi)*np.arccos(np.exp(-ff))
82
83      return F_tip
84
85  def _hub_correction(Nb, r, R, phi, R_hub=0.0):
86      """ Calculates a correction for the hub according to prandtls
            law
87
88      Args:
89          Nb (int): number of blades
90          r (float): radius of the section (m)
91          R (float): radius of the rotor (m)
92          phi (float): inflow angle (rad/s)
93
94      Returns:
95          float: hub correction coefficient
96      """
97      fh = 0.5 * Nb * (r - R_hub)
98      fh /= (r * np.sin(phi))
99      F_hub = (2/np.pi)*np.arccos(np.exp(-fh))
100
101     return F_hub
102
103 def algorithm_2(a:float, phi:float, radii, Nb:int,
104                 Ao, CLo, CDo, R_h=0.0, tip_corr=True,
                        hub_corr=True,
105                 toroidal=False):
106     """_summary_
107
108     Args:
109         a (float): axial induction factor
110         phi (float): inflow anfgle (rad/s)
111         radii (np.ndarray<float>): radii coords of the rotor (m)
112         Nb (int): number of blades
113         tip_corr (bool, optional): Turn on or off the Prandtl's
                tip correction
114         function. Defaults to True.
115
116     Returns:
117         dict: theta: induction angle (rad/s), chord: chord length
                (m)
118     """
119
120     N = len(radii)
121     R = radii[-1]
122     R_h = 0.00
123     theta = np.zeros(N)
124     c = np.zeros(N)
125
126     for i in range(N):
```

```
127         theta[i] = phi[i] - Ao
128 
129         Cn = CLo * np.cos(phi[i]) + CDo * np.sin(phi[i])
130 
131         c[i] = ((8*np.pi)/Nb) * (a[i]/(1-a[i])) * \
132                 ((np.sin(phi[i])*np.sin(phi[i]))/Cn) * radii[i]
133 
134         if tip_corr == 1:
135             t_coor = _tip_correction(Nb, radii[i], R, phi[i], R_h)
136             if np.isnan(t_coor) or t_coor == 0: t_coor = 2e-2
137             c[i] *= t_coor
138 
139 
140         if hub_corr == 1:
141             h_coor = _hub_correction(Nb, radii[i], R, phi[i], R_h)
142             if np.isnan(h_coor): h_coor = 1e-5
143             c[i] *= h_coor
144 
145         # print(c[i])
146     for i, t in enumerate(theta):
147         if t < 0.0: theta[i] = 0.0
148 
149 
150     return {"theta":theta, "chord": c}
151 
152 
153 def save_properties(V_0, OMEGA, R, NSEC, N_B, NACA, REYNOLDS,
    RADII, optm,
154                     i_hub, mean_chord, mean_a, mean_at,
                           mean_theta, reynolds_new,
155                     save_dir="./", file_name="BEM_results.txt"):
156 
157     """_summary_ Saves the properties of the BEM algorithm in a
        .txt file"""
158 
159     with open(f"{save_dir}/{file_name}", 'w') as f:
160         f.write("-------------------------------------------\n")
161         f.write("UNBEM ROTOR - LEA - ENM - FT - UnB\n")
162         f.write("-------------------------------------------\n")
163         f.write("Autor: Felipe Andrade\n")
164         f.write("Based on: Antonio Brasil Jr.\n")
165         f.write("===========================================\n")
166         f.write("BEM algorithm\n")
167         f.write("===========================================\n")
168         f.write("BEM INPUTS\n")
169         f.write("-------------------------------------------\n")
170         f.write(f"V_0: {V_0:.2f} m/s\n")
171         f.write(f"OMEGA: {OMEGA:.3f} rad/s,
               {(OMEGA*60)/(2*np.pi):.3f} rpm\n")
172         f.write(f"RADII: (R_0) 0.0 m, (R) {R:.3f} m, (NSEC) {NSEC}
               \n")
173         f.write(f"Diameter: {2*R:.3f} m\n")
```

```
174         f.write(f"lambda (TSR): {(OMEGA*R)/V_0:.1f}\n")
175         f.write(f"N_B: {N_B}\n")
176         f.write("-------------------------------------------\n")
177         f.write(f"NACA {NACA} polar data for Re:
                {reynolds_new:.2e}\n")
178         f.write("-------------------------------------------\n")
179         f.write(f"CLo: {optm['CLo']:.3f}\n")
180         f.write(f"CDo: {optm['CDo']:.3f}\n")
181         f.write(f"alpha_o: {optm['Ao']:.3f} rad,
                {np.degrees(optm['Ao'])} degrees\n")
182         f.write("-------------------------------------------\n")
183         f.write("BEM RESULTS\n")
184         f.write("-------------------------------------------\n")
185         f.write(f"First profile outside the hub - index:
                {i_hub}\n")
186         f.write(f"First profile outside the hub - radius:
                {RADII[i_hub]:.3f}m\n")
187         f.write(f"mean chord (c): {mean_chord:.3e} m\n")
188         f.write(f"mean axial induction (a): {mean_a:.3f}\n")
189         f.write(f"mean tang. induction (a'): {mean_at:.3f}\n")
190         f.write(f"half radius (R/2): {(R/2):.3e} m\n")
191         f.write(f"mean theta: {np.degrees(mean_theta):.3f}
                degrees\n")
192         f.write(f"-------------------------------------------\n")
193         f.write(f"Reynolds: {reynolds_new:.2e}\n")
194         f.write("===========================================")
195
196 class Bem:
197     """
198     Blade Element Momentum (BEM) algorithm implementation for wind
            turbine rotor design.
199
200     Parameters
201     ----------
202     TSR : float
203         Tip Speed Ratio (TSR) of the rotor.
204     V0 : float
205         Wind speed at hub height.
206     RADII : array_like
207         Array of radial positions of each section of the rotor.
208     N_B : int
209         Number of blades of the rotor.
210     NACA : int
211         NACA airfoil code of the blade sections.
212     REYNOLDS : float
213         Reynolds number of the flow.
214     TIP_CORR : bool, optional
215         Flag to enable tip loss correction. Default is 0 (no
                correction).
216     xfoil_path : str, optional
217         Path to the directory where the airfoil polar data files
                are located. Default is "./xfoil".
```

```
218     I_HUB : int, optional
219         Index of the first section outside the hub. Default is 0.
220 
221     Attributes
222     ----------
223     a : ndarray
224         Axial induction factor of each section of the rotor.
225     at : ndarray
226         Tangential induction factor of each section of the rotor.
227     phi : ndarray
228         Flow angle at each section of the rotor.
229     theta : ndarray
230         Twist angle of each section of the rotor.
231     chord : ndarray
232         Chord length of each section of the rotor.
233     Ao : float
234         Angle of attack at which the lift coefficient is maximum.
235     CLo : float
236         Lift coefficient at zero angle of attack.
237     CDo : float
238         Drag coefficient at zero angle of attack.
239     I_HUB : int
240         Index of the first section outside the hub.
241     mean_a : float
242         Mean axial induction factor of the rotor.
243     mean_at : float
244         Mean tangential induction factor of the rotor.
245     mean_chord : float
246         Mean chord length of the rotor.
247     mean_theta : float
248         Mean twist angle of the rotor.
249 
250     Methods
251     -------
252     run()
253         Runs the BEM algorithm.
254     calc_mean_props()
255         Calculates the mean properties of the rotor.
256     write(save_dir="./", file_name="BEM_results.txt")
257         Writes the results of the BEM algorithm to a text file.
258     write_latex_tables(save_dir="./", file_name="BEM_tables.tex")
259         Writes the results of the BEM algorithm to a LaTeX table.
260     """
261 
262     def __init__(self, TSR, V0, RADII, N_B, NACA, REYNOLDS,
263                  TIP_CORR=0, xfoil_path=None, I_HUB=0,
                        toroidal=False):
264 
265         if xfoil_path is None:
266             xfoil_path = "./xfoil"
267             if not os.path.exists(xfoil_path):
268                 os.mkdir(xfoil_path)
```

```
269
270         self.__xfoil_path = xfoil_path
271         self.toroidal = toroidal
272         self.__TSR = TSR
273         self.__RHUB = RADII[I_HUB]
274         self.__R = RADII[-1]
275         self.__V0 = V0
276         self.__OMEGA = TSR*V0/RADII[-1]
277         self.__RADII = RADII
278         self.__NSEC = len(RADII)
279         self.__N_B = N_B
280         self.__NACA = NACA
281         self.__REYNOLDS = REYNOLDS
282         self.__TIP_CORR = TIP_CORR
283
284         self.a = np.zeros(self.__NSEC)
285         self.at = np.zeros(self.__NSEC)
286         self.phi = np.zeros(self.__NSEC)
287         self.theta = np.zeros(self.__NSEC)
288         self.chord = np.zeros(self.__NSEC)
289         self.Ao = 0.0
290         self.CLo = 0.0
291         self.CDo = 0.0
292
293         self.I_HUB = I_HUB
294
295         self.mean_a = 0.0
296         self.mean_at = 0.0
297         self.mean_chord = 0.0
298         self.mean_theta = 0.0
299
300         self.run()
301         self.calc_mean_props()
302         np.savetxt("phi.txt",
303             np.degrees(np.transpose(np.array([self.phi,
304                 np.ones(self.__NSEC)*self.Ao,
305                 self.theta]))))
306
307     def run(self):
308         """
309         Runs the BEM algorithm.
310         """
311         alg_1 = algorithm_1(self.__TSR,self.__RADII,
                toroidal=self.toroidal)
312         self.a = alg_1["a"]
313         self.at = alg_1["at"]
314         self.phi = alg_1["phi"]
315         optm = get_optimal(self.__xfoil_path+
316                     f"/{self.__NACA}_{self.__REYNOLDS:.2e}.txt")
317         self.Ao = optm["Ao"]
318         self.CLo = optm["CLo"]
319         self.CDo = optm["CDo"]
```

```
320         args = [alg_1["a"], alg_1["phi"], self.__RADII, self.__N_B,
321         optm["Ao"], optm["CLo"], optm["CDo"]]
322         alg_2 = algorithm_2(*args, tip_corr=self.__TIP_CORR,
323                             toroidal=self.toroidal)
324         self.theta = alg_2["theta"]
325         self.chord = alg_2["chord"]
326 
327     def calc_mean_props(self):
328         """
329         Calculates the mean properties of the rotor.
330         """
331         self.mean_a = np.mean(self.a[self.I_HUB:])
332         self.mean_at = np.mean(self.at[self.I_HUB:])
333         self.mean_chord = np.mean(self.chord[self.I_HUB:])
334         self.mean_theta = np.mean(self.theta[self.I_HUB:])
335 
336     def write(self, save_dir=None, file_name="BEM_results.txt"):
337         """
338         Writes the results of the BEM algorithm to a text file.
339 
340         Parameters
341         ----------
342         save_dir : str, optional
343             Directory where the file will be saved. Default is
                "./".
344         file_name : str, optional
345             Name of the file. Default is "BEM_results.txt".
346 
347         Returns
348         -------
349         int
350             Returns 0 if the file was successfully written.
351         """
352         if save_dir is None:
353             save_dir = "./results"
354             if not os.path.exists(save_dir):
355                 os.mkdir(save_dir)
356 
357         with open(f"{save_dir}/{file_name}", 'w') as f:
358             f.write("-------------------------------------------\n")
359             f.write("UNBEM ROTOR - LEA - ENM - FT - UnB\n")
360             f.write("-------------------------------------------\n")
361             f.write("Autor: Felipe Andrade\n")
362             f.write("Based on: Antonio Brasil Jr.\n")
363             f.write("===========================================\n")
364             f.write("BEM algorithm\n")
365             f.write("===========================================\n")
366             f.write("BEM INPUTS\n")
367             f.write("-------------------------------------------\n")
368             f.write(f"V_0: {self.__V0:.2f} m/s\n")
369             f.write(f"OMEGA: {self.__OMEGA:.3f} rad/s,
                    {(self.__OMEGA*60)/(2*np.pi):.3f} rpm\n")
```

```
370             f.write(f"RADII: (R_0) {self.__RHUB:.3f} m,"
371                     +f" (R) {self.__R:.3f} m, (NSEC) {self.__NSEC}
                           \n")
372             f.write(f"Diameter (D): {2*self.__R:.3f} m\n")
373             f.write(f"lambda (TSR): {self.__TSR:.1f}\n")
374             f.write(f"N_B: {self.__N_B}\n")
375             f.write("-----------------------------------------\n")
376             f.write(f"NACA {self.__NACA} polar data for Re:
                   {self.__REYNOLDS:.2e}\n")
377             f.write("-----------------------------------------\n")
378             f.write(f"CLo: {self.CLo:.3f}\n")
379             f.write(f"CDo: {self.CDo:.3f}\n")
380             f.write(f"alpha_o: {self.Ao:.3f} rad,
                   {np.degrees(self.Ao)} degrees\n")
381             f.write("-----------------------------------------\n")
382             f.write("BEM RESULTS\n")
383             f.write("-----------------------------------------\n")
384             if self.I_HUB != 0:
385                 f.write(f"First profile outside the hub - index:
                       {self.I_HUB}\n")
386                 f.write(f"First profile outside the hub - radius:
                       {self.I_HUB:.3f}m\n")
387             f.write(f"mean chord (c): {self.mean_chord:.3e} m\n")
388             f.write(f"mean axial induction (a):
                   {self.mean_a:.3f}\n")
389             f.write(f"mean tang. induction (a'):
                   {self.mean_at:.3f}\n")
390             f.write(f"half radius (R/2): {(self.__R/2):.3e} m\n")
391             f.write(f"mean theta:
                   {np.degrees(self.mean_theta):.3f} degrees\n")
392             f.write(f"-----------------------------------------\n")
393             f.write(f"Reynolds: {self.__REYNOLDS:.2e}\n")
394             f.write("=========================================")
395 
396             return 0
397 
398     def write_latex_tables(self, save_dir=None,
           file_name="BEM_tables.tex"):
399         """
400         Writes the results of the BEM algorithm to a LaTeX table.
401 
402         Parameters
403         ----------
404         save_dir : str, optional
405             Directory where the file will be saved. Default is
                   "./".
406         file_name : str, optional
407             Name of the file. Default is "BEM_tables.tex".
408 
409         Returns
410         -------
411         int
```

```
412             Returns 0 if the file was successfully written.
413         """
414         if save_dir is None:
415             save_dir = "./results"
416             if not os.path.exists(save_dir):
417                 os.mkdir(save_dir)
418         with open(f"{save_dir}/{file_name}", 'w') as f:
419             # Write inputs table
420             f.write("\\begin{table}[ht]\n")
421             f.write("\\centering\n")
422             f.write("\\caption{BEM inputs}\n")
423             f.write("\\begin{tabular}{|l|l|}\n")
424             f.write("\\hline\n")
425             f.write("Parameter & Value \\\\\n")
426             f.write("\\hline\n")
427             f.write(f"TSR & {self.__TSR:.2f} \\\\\n")
428             f.write(f"Inflow velocity (m/s) & {self.__V0:.2f}
                    \\\\\n")
429             f.write(f"Rotor radius (m) & {self.__R:.2f} \\\\\n")
430             f.write(f"Number of sections & {self.__NSEC} \\\\\n")
431             f.write(f"Number of blades & {self.__N_B} \\\\\n")
432             # f.write(f"NACA airfoil & {self.__NACA} \\\\\n")
433             # f.write(f"Reynolds number & {self.__REYNOLDS:.2e}
                    \\\\\n")
434             f.write("\\hline\n")
435             f.write("\\end{tabular}\n")
436             f.write("\\end{table}\n\n")
437 
438             # Write outputs table
439             f.write("\\begin{table}[ht]\n")
440             f.write("\\centering\n")
441             f.write("\\caption{BEM outputs}\n")
442             f.write("\\begin{tabular}{|l|l|}\n")
443             f.write("\\hline\n")
444             f.write("Parameter & Value \\\\\n")
445             f.write("\\hline\n")
446             f.write(r"Mean axial induction $\\bar{a}$"
447                     +f" & {self.mean_a:.3f} \\\\\n")
448             f.write(r"Mean tangential induction $\bar{a^\prime}$"
449                     +f" & {self.mean_at:.3f} \\\\\n")
450             f.write(r"Mean chord length $\bar{c}$ (m)"
451                     +f" & {self.mean_chord:.3e} \\\\\n")
452             f.write(r"Mean induction angle $\bar{\theta}$
                    (degrees)"
453                     +f" & {np.degrees(self.mean_theta):.3f}
                        \\\\\n")
454             f.write("\\hline\n")
455             f.write("\\end{tabular}\n")
456             f.write("\\end{table}\n")
457 
458         return 0
459 
```

```
460
461 if __name__ == "__main__":
462     TSR = 2.8
463     V0 = 8.0
464     RADII = np.linspace(0.0, 1.1, 100)
465     N_B = 4
466     NACA = "0015"
467     REYNOLDS = 3.7596e+06
468     BEM = Bem(TSR, V0, RADII, N_B, NACA, REYNOLDS)
469     BEM.write()
470     BEM.write_latex_tables()
```

Appendix B – OpenFOAM Files

B.1 Numerical Schemes

Code B.1 – fvSchemes

```
/*--------------------------------*- C++
   -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  11
     \\/     M anipulation  |
\*---------------------------------------------------------------------------
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   * * * * * //

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    /* default         Gauss linear; */
    default         cellMDLimited Gauss linear 0.5;
    // limited         cellLimited Gauss linear 0.5;
    limited         cellLimited Gauss linear 1.0;
    grad(U)         $limited;
    grad(k)         $limited;
    grad(omega)     $limited;
    grad(epsilon)   $limited;
}

divSchemes
{
    default             none;
    div(phi,U)          Gauss linearUpwind grad(U);
    // div(phi,U)          Gauss upwind;
    div(phi,omega)      Gauss upwind;
    div(phi,k)          Gauss upwind;
```

```
41      div(phi,epsilon)    Gauss upwind;
42      div((nuEff*dev2(T(grad(U))))) Gauss linear;
43  }
44
45  laplacianSchemes
46  {
47      default         Gauss linear limited corrected 0.333;
48      // default         Gauss linear limited corrected 0.777;
49  }
50
51  interpolationSchemes
52  {
53      default         linear;
54  }
55
56  snGradSchemes
57  {
58       default         limited corrected 0.333;
59      // default         limited corrected 0.777;
60  }
61
62  wallDist
63  {
64      method meshWave;
65  }
66
67
68  //
       ************************************************************************
       //
```

B.2 Numerical Solution Setup

Code B.2 – fvSolutions

```
1   /*--------------------------------*- C++
       -*----------------------------------*\
2     =========                 |
3     \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4      \\    /   O peration     | Website:  https://openfoam.org
5       \\  /    A nd           | Version:  11
6        \\/     M anipulation  |
7   \*---------------------------------------------------------------------------
8   FoamFile
9   {
10      format      ascii;
11      class       dictionary;
12      object      fvSolution;
13  }
```

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

solvers
{
    p
    {
        solver          PBiCGStab;
        // smoother        DICGaussSeidel;
        preconditioner  DIC;
        tolerance       1e-6;
        relTol          0.01;
        cacheAgglomeration no;
        minIter         15;
        maxIter         150;
    }

    pFinal
    {
        solver GAMG;
        tolerance 1e-06;
        relTol 0.0;
        smoother GaussSeidel;
        nPreSweeps 0;
        nPostSweeps 2;
        cacheAgglomeration on;
        agglomerator faceAreaPair;
        nCellsInCoarsestLevel 1000;
        mergeLevels 1;
    }

    // pFinal
    // {
    //      $p;
    //      nPreSweeps 0;
    //      nPostSweeps 2;
    //      cacheAgglomeration on;
    //      agglomerator faceAreaPair;
    //      nCellsInCoarsestLevel 1000;
    //      mergeLevels     1;
    //      relTol          0.0;
    // }

    "pcorr.*"
    {
        $p;
        tolerance       1e-2;
        relTol          0;
    }

    MeshPhi
    {
```

```
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-2;
        relTol          0.0;
    }

    "(U|k|epsilon|omega)"
    {
        solver PBiCGStab;
        preconditioner DILU;
        minIter         4;
        tolerance       1e-8;
        relTol          0.0;
    }

    "(U|k|epsilon|omega)Final"
    {
        $U;
        relTol          0;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 2;
    consistent yes;
}

potentialFlow
{
    nNonOrthogonalCorrectors 10;
}

PIMPLE
{
    momentumPredictor   on;
    correctPhi          yes;
    correctMeshPhi      yes;
    nOuterCorrectors    1;
    nCorrectors         3;
    nNonOrthogonalCorrectors 1;

    residualControl
    {
        p               2.5e-6;
        U               1e-6;
        k               4e-6;
        epsilon         1e-6;
        omega           2e-6;
    }
}

```

```
117 relaxationFactors
118 {
119     fields
120     {
121         p               0.3;
122     }
123     equations
124     {
125         U               0.7;
126         k               0.7;
127         omega           0.7;
128         epsilon         0.7;
129     }
130 }
131
132 cache
133 {
134     grad(U);
135 }
136
137 // ************************************************************************* //
```