Politecnico di Torino

Facoltà di Ingegneria

Corso di laurea in Ingegneria Energetica e Nucleare

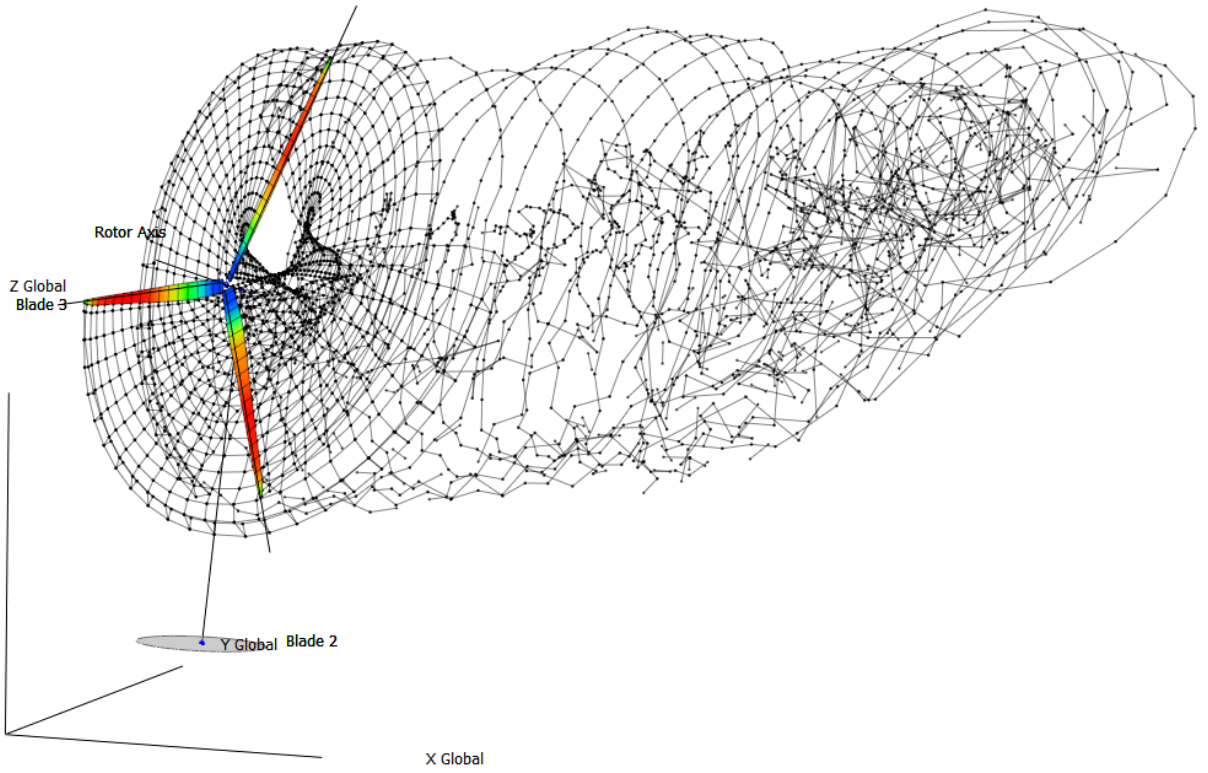
Master Thesis

Integration of a 6DOF floating platform model in *QBlade*

Supervisor
Ing. Giovanni Bracco
Advisors
Prof. Giuliana Mattiazzo

Candidate:
Michele Riccioli

March 2019


Rotor Axis
Z Global
Blade 3
Y Global
Blade 2
X Global

Abstract

This work was done in cooperation with the wind energy group at the Department of Experimental Fluid Mechanics of TU Berlin, led by Prof. Dr. Christian Oliver Paschereit. They have developed a software, *QBlade*, an open-source wind turbine calculation software, distributed under the GNU General Public License. The software is integrated into XFOIL, an aerofoil design and analysis tool[1]. Its purpose is the design and aerodynamic simulation of wind turbines. The integration in XFOIL allows for the user to rapidly design custom aerofoils and compute their performance curves, extrapolating the performance data to a range of 360° angle of attack, and directly integrate them into a wind turbine rotor simulation. The idea was to integrate a floating platform model into *QBlade*, to make it able to simulate also floating offshore wind turbines, creating a graphical interface to make the user able to choose a particular wave and change some parameters, such as the characteristics of the mooring lines and the initial position of the system, obtaining results in the post-processing tool of the software.

The thesis is organized in such a way to provide a view of the mathematical models used to create the *Floater Module* and the one used to model the wind turbine by the external software, together with the procedure followed to integrate the *Floater Module* in *QBlade*. After the integration, some test cases are carried on simulating the behaviour of the system under different operating and not operating conditions, to verify a correct coupling between the two software, and a comparison between the offshore and onshore installations is done. For this reason, a floating offshore wind turbine system is presented, together with the characteristics which are useful to simulate its behaviour in the *floater module*.

[1] `http://web.mit.edu/drela/Public/web/xfoil/`

Contents

List of Figures

List of Tables

Acknowledgements

Finally, here we are at the end of the road. When I decided to start this adventure, I was little more than a kid, with so many dreams and a great desire to do, and a lot of fear for what was waiting for me. Now, I think back to what I have left behind, to the experiences I have lived, to the people I met along the way, and I am convinced that my decision to be "up in Turin" has brought only and only to an improvement in my way of thinking, of living, of acting.

I therefore thank my parents first. Thank you for all you have done, do and will continue to do for me. You were the safe haven in which to dock in the darkest moments, two pillars without which I surely would have yielded. Definitely, the smartest people I've ever met in my life.

I thank all my family for their constant presence and all the shared moments. One of the best things to do every time I came home was to come and hug you one by one. I thank Elena for being the most special sister I ever could have wished for, and for always believing in my abilities. Your smile and your way of being are something unique, never forget that I'll always be there for you.

A thought to my niece Nicole, that life smiles at you and brings as many satisfactions as possible.

I thank Giulia, for the blind trust she has placed in me and for the support given to me in these turbulent months. You are a really smart girl, who in a short time has already been able to give me so much and be appreciated.

A special place in these thanks for my friends Simone and Lorenzo. We met by chance, and a deep relationship was born between us: as if we had known each other as children. Without you it would not have been the same, you are an important piece of my life. I will never forget everything we have shared over these years.

Finally, thanks to my grandfather. I know how much you would have liked to be there in person in this moment of my life, I always remember all the stories you invented for me when I was a child, and the esteem you had for me. You have been a solid rock and an example: with a few words, a smile, a look, you knew how to put everything back in place. *I dedicate to you this work.*

Ringraziamenti

Finalmente, eccoci alla fine della strada. Quando decisi di iniziare questa avventura, ero poco più di un ragazzino, con tanti sogni e tanta voglia di fare, e molta paura per quello che mi aspettava. Adesso, ripenso a quello che mi sono ormai lasciato alle spalle, alle esperienze vissute, alle persone che ho conosciuto lungo il tragitto, e mi convinco del fatto che questa mia scelta di venir "*su a Torino*" abbia portato solo e soltanto ad un miglioramento nel mio modo di pensare, di vivere, di agire.

Ringrazio quindi, per primi, i miei genitori. Grazie per tutto quello che avete fatto, fate e continuerete a fare per me. Siete stati il porto sicuro in cui attraccare nei momenti più bui, due pilastri senza i quali sicuramente avrei ceduto. Decisamente, le persone più in gamba che abbia mai conosciuto nella mia vita.

Ringrazio tutta la mia famiglia per la costante presenza e tutti i momenti condivisi. Una delle cose più belle da fare ogni volta che sono tornato a casa è stata venirvi a riabbracciare uno per uno.

Ringrazio Elena per essere la sorella più speciale che abbia mai potuto desiderare, e per aver sempre creduto nelle mie capacità. Il tuo sorriso e il tuo modo di essere sono un qualcosa di unico, non dimenticare mai che ci sarò sempre per te.

Un pensiero alla mia nipotina Nicole, che la vita ti sorrida e porti quante più soddisfazioni possibili.

Ringrazio Giulia, per la cieca fiducia che ripone in me e per il sostegno datomi in questi mesi così turbolenti. Sei una ragazza davvero in gamba, che in poco tempo ha saputo già donarmi tanto e farsi apprezzare.

Un posto speciale in questi ringraziamenti per i miei amici Simone e Lorenzo. Ci siamo conosciuti per caso, e tra di noi è nato un rapporto profondo: come se ci fossimo conosciuti da bambini. Senza di voi non sarebbe stato lo stesso, siete un pezzo importante della mia vita. Non dimenticherò mai tutto quello che abbiamo condiviso in questi anni.

Infine un grazie a mio nonno. So quanto avresti voluto esserci di persona in questo momento della mia vita, ripenso sempre a tutti i racconti che inventavi per me quando ero bambino, e alla stima che avevi nei miei confronti. Sei stato una roccia solida e un esempio: con poche parole, un sorriso, uno sguardo, sapevi rimettere tutto a posto. *E' a te che dedico questo lavoro.*

Introduction

In our world, climate changes are leading to new challenges for the energy sector. As stated by the Climate Science Special Report [1], many lines of evidence demonstrate that human activities, especially emissions of greenhouse gases, are primarily responsible for the observed climate changes in the industrial era, especially over the last six decades.

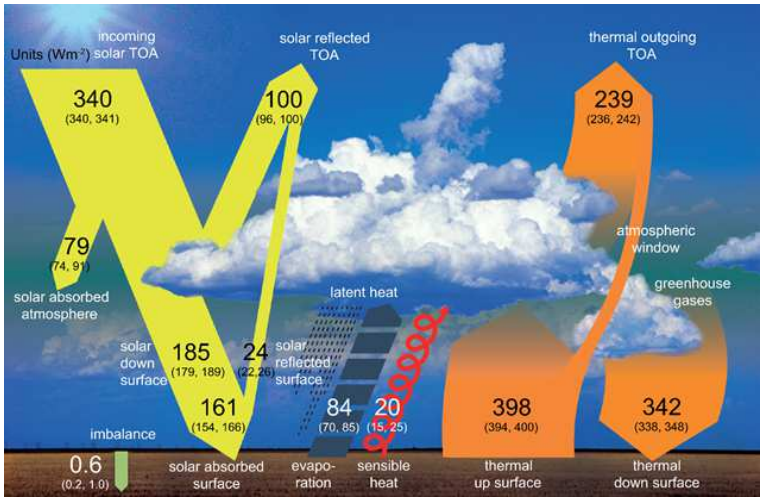


Figure 1: *Earth's thermal balance*

The temperature of the Earth system is determined by the amounts of incoming (short-wavelength) and outgoing (both short- and long-wavelength) radiation.
In the modern era, radiative fluxes are measured by means of satellites. About a third (29.4%) of incoming, short-wavelength energy from the sun is reflected back to space, and the remainder is absorbed by Earth's system, as illustrated in Figure 1. The fraction of sunlight scattered back to space is determined by the reflectivity (albedo) of clouds, land surfaces (including snow and ice), oceans, and particles in the atmosphere. The amount and albedo of clouds, snow and ice covers are particularly strong determinants of the amount of sunlight reflected back to space because their albedo are much higher than that of land and oceans. In addition to reflected sunlight, Earth loses energy through infrared (long-wavelength) radiation from the surface and the atmosphere. Absorption by greenhouse gases (GHGs) of infrared energy radiated from the surface leads to warming of the surface and atmosphere, decreasing the thermal flux from the Earth to Space. One of the main consequences is

the rising of the Global Surface Temperature: Earth's temperature is generally variable, due to natural phenomena, but in the last decades the human influence has started to be the most important driving force for the temperature rising, as shown in Figure 2. Evidence continues to mount for an expansion of the tropics over the past several decades, with a poleward expansion of the Hadley cell and an associated poleward shift of the subtropical dry zones and storm tracks in each hemisphere. The rate of expansion is uncertain and depends on the metrics and data sources that are used. Recent estimates of the widening of the global tropics for the period 1979–2009 range between 1° and 3° latitude (between about 70 and 200 miles) in each hemisphere, an average trend of approximately 0.5° and 1.0° per decade. While the roles of increasing greenhouse gases in both hemispheres, stratospheric ozone depletion in the Southern Hemisphere, and anthropogenic aerosols in the Northern Hemisphere, have been implicated as contributors to the observed expansion, there is uncertainty in the relative contributions of natural and anthropogenic factors, and natural variability may currently be dominating.

However, changes in precipitation are one of the most important potential outcomes of a warming world because precipitation is integral to the very nature of society and ecosystems. These systems have developed and adapted to the past envelope of precipitation variations. Any large changes beyond the historical envelope may have profound societal and ecological impacts. Climate changes in Alaska and across the Arctic continue to outpace changes occurring across the globe. The Arctic, defined as the area north of the Arctic Circle, is a vulnerable and complex system integral to Earth's climate. The vulnerability stems in part from the extensive cover of ice and snow, where the freezing point marks a critical threshold that when crossed has the potential to transform the region. Because of its high sensitivity to radiative forcing and its role in amplifying warming, the arctic cryosphere is a key indicator of the global climate state. Accelerated melting of multiyear sea ice, mass loss from the Greenland Ice Sheet, reduction of terrestrial snow cover, and permafrost degradation are stark examples of the rapid Arctic-wide response to global warming. These local arctic changes influence global sea level, ocean salinity, the carbon cycle, and potentially atmospheric and oceanic circulation patterns. Arctic climate change has altered the global climate in the past and will influence climate in the future.

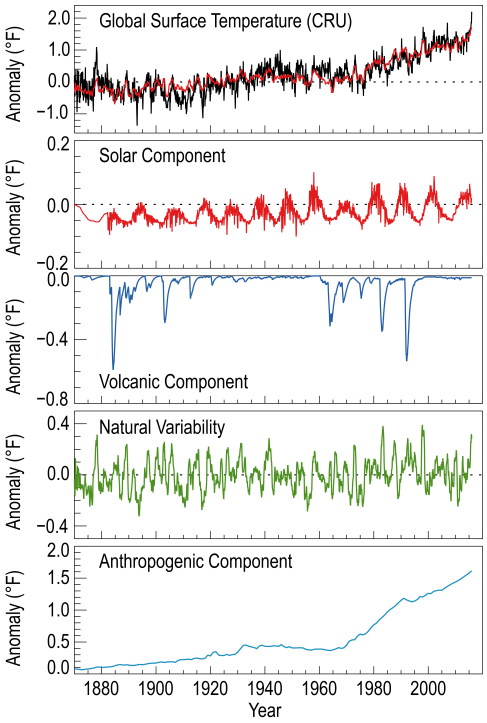


Figure 2: *Contributions to the rise of Global Surface Temperature*

Due to all these reasons, and many others, humanity has to start going in a different direction, trying to decarbonize its activities, and in particular the power generation sector, that continues to be based on fossil fuels. The challenge that our generation has to afford is probably the biggest humanity has ever seen: unfortunately, capitalism continue to influence all human activities, and often a cheaper solution is preferred instead of a more efficient, cleaner one. But as we continue speaking about the changes that our planet is undergoing, a lot of us continue acting like the only day to live is today. Future generations will have to deal with problems that seem to be far, but tomorrow will became more and more serious. Our task is to try to help them, starting to introduce a new mentality, based on the respect of all we can find above and under our heads. Our mission is to build a new world, a new society, a new kind of humanity. No matter if our task is difficult, no matter if it will take a lot of time, no matter if we have to use all our mental and physical forces; all these efforts will, one day, be repaid.

I hope that these wishes will, in future, become the wishes of all.

Chapter 1

Offshore Wind Power

1.1 Brief History and Economical Aspects

The first offshore wind farm was realized in Denmark in 1991 and was known as the Vindeby Offshore Wind Farm. Consisting of eleven 450 kW turbines, was built at a near-shore site. This and the projects that followed last century generally used concrete foundations and small turbines, similar to the ones used by the onshore installations. In 2002 the first utility-scale offshore wind farm, with a capacity of 160 MW (80 turbines of 2 MW each), was grid-connected at Horns Rev, off the coast of Denmark. Starting from that point to the end of 2015, global operating capacity grew from 0,26 GW to 12,7 GW.

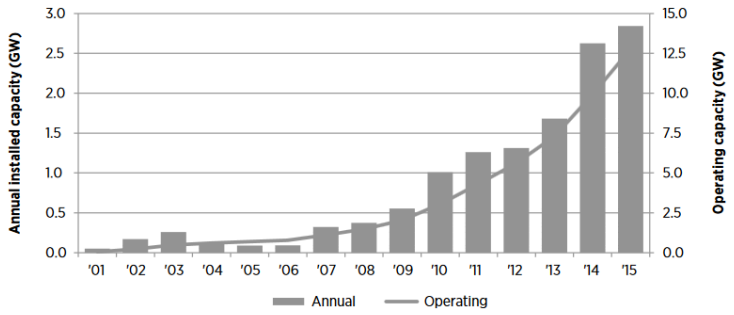


Figure 1.1: *Global annual installed capacity and operating capacity of offshore wind, 2001-2015. Taken from [2]*

This growth was driven by enabling policies and attractive incentives to improve energy security of supply and build an industry [2], coupled with the growing interest of the constructors in producing new types of machines. Early projects were funded primarily by large utility companies, supported by attractive public incentives in the form of fixed, regulated prices. Towards the end of this period, more projects were funded by third-party debt, with lower levels of public financial support. From 2001 through 2015, on average, projects moved to sites farther from shore, in deeper waters and with higher wind speeds. The rated power of turbines increased and manufacturers de-

veloped new wind turbines specifically for the offshore market. Installation methods and offshore structures became more cheaper and functional. Costly, specialised components gradually were replaced by more affordable standard components, produced in series.

Generally, water depths, distances to ports and project sizes increased as the industry moved from early commercial projects mainly in UK waters to a wider geographical spread of projects. Installation in German waters accelerated these increases towards the end of the period.

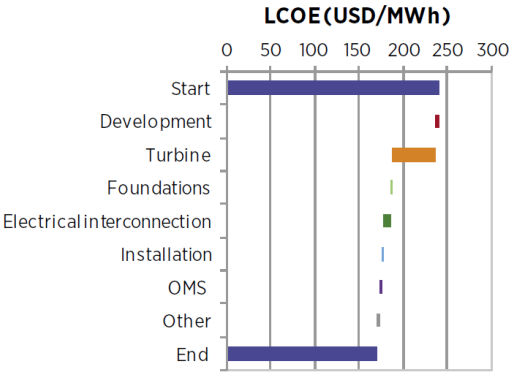


Figure 1.2: *LCOE reduction by technology element for projects commissioned in 2001-2015. Each bar represents the impact of innovations in a given element. Taken from [2]*

The LCOE of a typical offshore wind farm commissioned in 2015 is about 170 USD/MWh on average, while in 2001 it was about 240 USD/MWh. In Figure 1.2 is represented the LCOE's reduction thanks to innovations in development, turbines, foundations, installation, OMS and other non-technology factors. The most important innovation was the introduction of a new generation of wind turbines, with very large rotors, and a range of innovations in foundations construction. "Other" changes in cost came from financing costs and other non-technological issues, such as project life, competition, exchange rates and commodity prices. The composition of the LCOE in 2015 can be seen in Figure 1.3. Nearly half of the cost is attributable to financing, related to the interests on the debt and the rate of return on the equity needed to fund these projects: this means that a very important objective is the reduction of the risk associated to the technology.

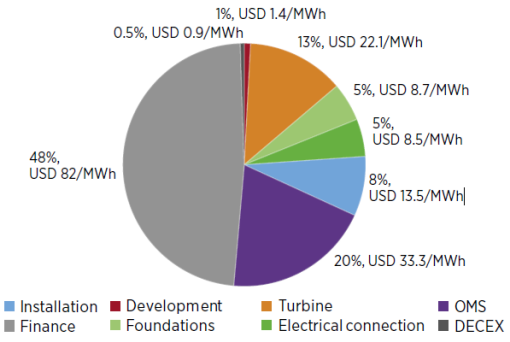


Figure 1.3: *Contribution of each element to cost of energy for typical project commissioned, end-2015. Taken from [2]*

The industry of the FOWTs has seen a great acceleration in the last years, thanks to huge investments, but the LCOE of this kind of systems is still below

the one of the bottom-fixed wind turbines. Carbon Trust [4] estimates that the potential for competitiveness of floating turbines is 130 €/MWh for commercial deployments, and as low as 110 €/MWh for leading concepts. Actually, the bottom-fixed commercial projects in the UK are on track with the 2020 target of 130 €/MWh, but to be able to compete with this kind of systems, the FOWTs must have a steeper curve of improvement. The main focus now is trying to de-risk , via better design tools and prototype deployments.

Actually, as stated by [4], the offshore wind market is dominated by countries with shallow water depths (<50m), that permit to install fixed-bottom structures. However, there is extensive wind resource in deep waters (50-200m), with a significant potential for growth in a great number of European countries, mainly in the North of Europe, off the coast of Scotland and England. EU targets for offshore wind of 40GW by 2020 and 150GW by 2030 seems to be achievable using prevalently fixed-bottom foundations. In all the cases, accessing deeper water sites could permit to reach 460GW of installed power by 2050. It is noticeable that, while cost data for fixed-bottom foundations are numerous, the same can't be said for floating foundations: cost for new and innovative technologies tends to increase during conception, while, during the Optimisation and Industrialisation phases it tends to decrease reaching an asymptotical value, as shown in Figure 1.4.

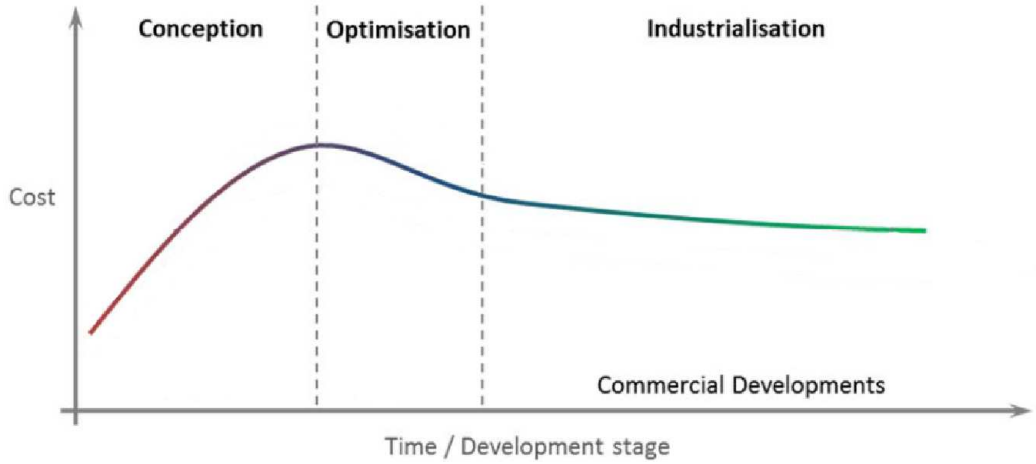


Figure 1.4: *Technology cost evolution through time*

It is also interesting to compare the actual LCOE of FOWTs, with the LCOE of other conventional and unconventional energy sources Figure 1.5.

It can be noticed that the LCOE of FOWTs (around 130 €/MWh) is competitive with other unconventional energy sources, such as Solar Thermal Towers and Fuel Cells.

Figure 1.5: *LCOE comparison for different conventional and unconventional energy sources. Taken from [5]*

1.2 Technology Readiness

Technology Readiness Level (TRL) is a key indicator for how advanced a technology is, considering as final objective its commercialisation, as reported by [4]. The indicator can assume seven different values, reported in

Currently several concepts have already reached the level of scaled tests in wave tanks or offshore sites (TRL 4-5) and are moving towards the full scale demonstration.

0. Unproven concept	Idea/preliminary study/patent
1. Proven concept	Desk-based basic design assessment/proof of concept
2. Validated concept	Detailed numerical modelling/structural assessment
3. Prototype tested	scaled testing (<1 MW demonstration)
4. Environment tested	Offshore demonstration 1-5 MW turbine
5. System tested	Full-scale demonstration with >5 MW turbine
6. System installed	Full-scale demonstration with >5 MW turbine with >1 year operation
7. Field proven	Commercial project

Table 1.1: *TRL and meaning of each level*

Figure 1.6: *Archetypes of floating foundations adopted by the wind energy industry. From left to right: spar buoy, semi-submersible platform and tension-leg platform.*

1.2.1 Floater types

The floating foundation types adopted by the wind energy sector can mainly be described as one of the following types [3]:

- *Spar buoy types*: the center of mass is moved far below the center of buoyancy, creating a strong restoring moment when the structure is pushed out of equilibrium. The simple structure of the spar-buoy is tipically fairy easy to fabricate, providing good stability. However, the large draft requirement can create logistical challenges during assembly, transportation and installation, constraining deployment in waters with a depth greater than 100m.

- *Semi-submersible Platform (Semi-Sub)*: uses the hydrostatic stiffness of substructures piercing the water plane area in positions offset from the center-axis of the tower. This provides the restoring stiffness needed for the overall stability. Often requires a large and heavy structure to maintain a good stability, but a low draft allow for more flexible application and simpler installation.

- *Tension-Leg Platform (TLP)/Tension-Leg Buoy(TLB)*: utilize the mooring in actively keeping the structure upright: this kind of structure is inherently unstable, because the center of gravity is above the center of buoyancy. so the mooring provides the righting moment by being connected to off-axis fairleads, and hereby transfers the wind turbine loads into the anchors below. The shallow draft and tension stability allows for a smaller and lighter structure, but this design increases stresses on the tendons and anchors system.

1.2.2 Mooring and anchoring systems

Each type of floater uses a particular mooring/anchoring system. Here are summarized the main characteristic of each one of it.

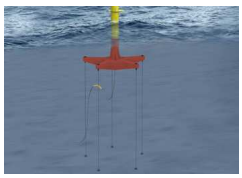
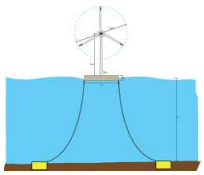
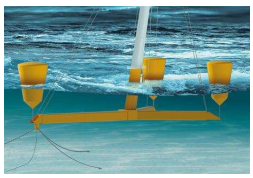
	Taut-leg	Catenary	Semi-taut
			
Material	Synthetic fibres/wires which use the buoyancy to maintain high tension	Long steel chains/wires whose weight and shape holds the platform in place	Synthetic fibres/wires incorporated with a turret system. A single point on the floater is connected to a turret with several semi-taut mooring lines
Loading at anchoring point	Vertical loading	Horizontal loading	45° loading
Loading on the anchors	Large loads	Long mooring lines reduce loads on anchors	Medium loads
Horizontal movements	Very limited	Some degree	Limited, the full structure can swivel around the turret connection
Wave-induced motion	High tension limits motion (pitch/roll/heave)	Weight of mooring lines limits floater motion, greater freedom than taut leg	Susceptible, due to the presence of the single connection point
Installation	Challenging	Simple	Simple

Table 1.2: Mooring and anchoring systems

Regarding the anchor choice , it depends on the particular site considered for the installation. The choice is among the following types:

- *Drag-embedded:* suited to not too stiff, cohesive sediments, the type of loading is horizontal. The installation is quite easy, and the anchor can

be recovered during decommissioning.

- *Driven Pile:* applicable in a wide range of seabed conditions, the loading can be horizontal/vertical. Regarding the installation, it requires hammer piling.
- *Suction Pile:* not suited in loose sandy soils or stiff soils, where the penetration is difficult. The load applied can be vertical or horizontal. Its installation is simple and less invasive than other methods. It permits an easy removal during decommissioning.
- *Gravity Anchor:* requires medium to hard soil conditions, usually is vertically loaded. Due to its large size and weight it can increase the installation cost, and is difficult to remove during decommissioning.

Chapter 2

Floater Module Mathematical Model

2.1 Reference frames and degrees of freedom

To describe the movements of a FOWT, generally two right-handed reference frames are employed (see Figure 2.1):

- the LSA (Local System Axes), with origin the center of gravity of the whole structure, the x axis being aligned with the reference incoming wave direction and the z axis pointing upwards along the symmetry axis of the tower;
- the FRA (Fixed Reference Axes), with origin the intersection between the tower symmetry axis and the still water level line, the axes being parallel, when the system is at rest, with the ones of the LSA. This frame is fixed, and thus represents an inertial reference frame.

The system is modelled as a rigid body, and, to describe its kinematics, the motion of the LSA are described in the FRA by the vector of the three translations and rotations:

$$\vec{x} = (x, y, z, r_x, r_y, r_z) \tag{2.1}$$

In this work, all six degrees of freedom are taken into account:

- x: **surge**, longitudinal translation, positive backwards
- y: **sway**, lateral translation, positive to starboard side
- z: **heave**, vertical translation, positive upwards
- r_x: **roll**, rotation around the x axis, positive right turning
- r_y: **pitch**, rotation around the y axis, positive right turning
- r_z: **yaw**, rotation around the z axis, positive right turning

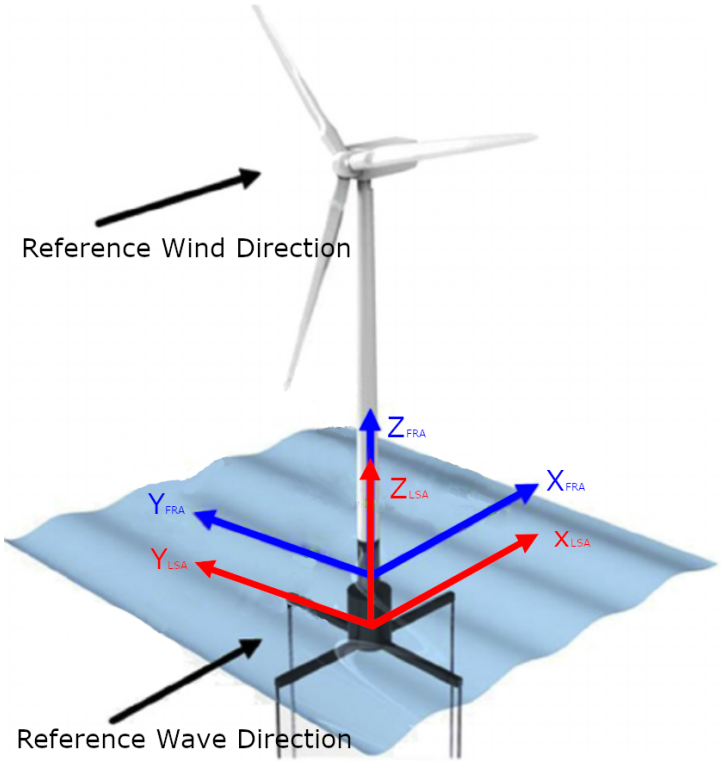


Figure 2.1: *Typical FOWT reference frame*

2.2 Floater

In this section, the mathematical model by which waves and hydrodynamic forces that they produce on the floater can obtained is described.

2.2.1 Waves

They are the source of excitation for the floating platform, together with wind. There are three main mathematical models to describe waves:

- the simplest possible model is the one of the **regular waves**, in which waves are modelled with a unique frequency and amplitude as sinusoidal functions;
- the **second order Stokes waves**, a potential flow solution expanded at the second order;
- the **irregular waves**, a superposition of infinite regular waves moving in different directions.

In this work only *irregular waves* are considered.

Modelling of irregular waves

Real waves are the result of the wind blowing on the water surface. *Swells* are long-crested nearly unidirectional sinusoidal waves, because they were generated by the wind in a previous space and time, but local winds still affect the

sea surface, producing short-crested multidirectional highly-irregular waves. For this reason, the sea surface can be seen as a superposition of *theoretically infinite* regular waves components with different height, frequency, wavelength and random phase using a Fourier series (Equation 2.3). For a single direction, we can write:

$$\eta(x,t) = \sum_{n=1}^{N} a_n sin(\omega t - k_n x + \varphi_n) \tag{2.2}$$

where the subscripts n indicates that the wave parameters are relative to the n-th component of the summation. The summation is considered to be finite because the contribution of the n-th wave becomes less significant for increasing frequencies. A wave spectral density function $S_\eta(\omega_n)$, calculated as the variance function of the waves amplitudes a_n, carries information about the significance of each wave component identified by the n-th frequency ω_n, as can be seen in Figure 2.2. A narrower curve represents waves close to be regular.

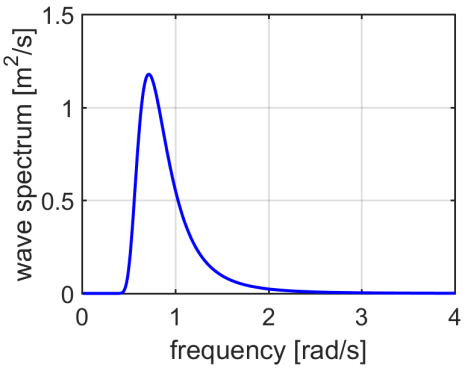


Figure 2.2: *Example of wave spectrum*

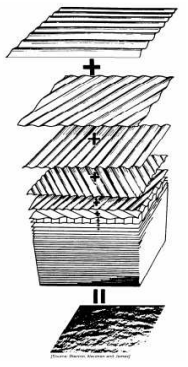


Figure 2.3: *Superposition of different regular waves*

An irregular wave is modelled using some statistical parameters:

- *Significant Wave Height* $H_{\frac{1}{3}} = H_s$: average height of the highest one third of the individual trough to crest heights in a wave record;

- *Average wave period* T_1;
- *Average zero-crossing wave period* $T_2 = \overline{T_z}$: average zero up-crossing of crests or throughs period;
- *Average wave height* $\overline{H}$;
- *Peak Wave Period* T_p: period corresponding to the frequency ω_p of the maximum value of the wave spectrum.

Standard wave spectra

Several standard-form description of wave spectra may be found in literature, but the most used are the *JONSWAP* and the *Pierson-Moskowitz* spectra, based on the significant wave height H_s and a reference wave period $\overline{T}$ that can be chosen between T_1, T_2 or T_p.

The **JONSWAP** (Joint North Sea Wave Project)spectrum [6], developed for the North Sea coastal wind-generated waves:

$$S_\eta(\omega) = \frac{320 H_s^2}{T_p^4}\omega^{-5}\gamma^A e^{\frac{-1950}{T_p^4}\omega^{-4}}$$

with:

$$\gamma = 3.3 \text{ peak enhancement factor}$$

$$A = \exp\left[-\left(\frac{\frac{\omega}{\omega_p}-1}{\sigma\sqrt{2}}\right)^2\right]$$

$$\omega_p = \frac{2\pi}{T_p}$$

$$\sigma = \begin{cases} 0.07 & \omega \geq \omega_p \\ 0.09 & \omega < \omega_p \end{cases}$$

The **Pierson-Moskowitz** or **Bretscheider** spectrum, well-suited for long-crested waves in open sea:

$$S_\eta(\omega) = \frac{173 H_s^2}{T_1^4}\omega^-5 e^{-\frac{692}{T_1^4}\omega^{-4}}$$

The two spectra are compared in Figure 2.4 for different peak periods T_p and same wave height H_s. It's worth noting that the JONSWAP spectrum has a more narrow distribution around the peak, whereas the Bretscheider spectrum is more distributed.

Transformation to time series

The greatest part of marine technology works in time domain. So, the statistical description of the wave using the spectra has to be transformed into a deterministic time history of wave elevation, avoiding to lose the statistical original properties. This can be done filling in all the constant of Equation 2.2, that are the amplitude a_n, the wave number k_n and the phase φ_n for each frequency component ω_n, chosen at evenly-spaced intervals $\Delta\omega_n$ along the frequency axis.

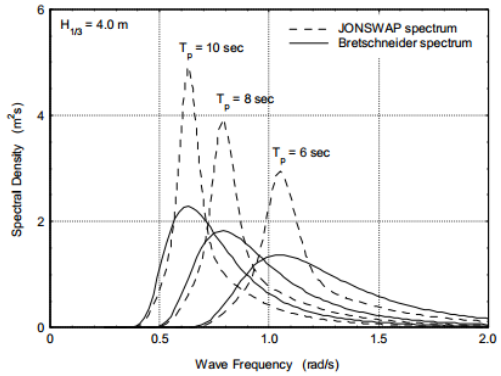


Figure 2.4: *JONSWAP and Bretschneider spectra*

- a_n is determined considering that the finite area under the relative $\Delta\omega_n$ interval of the spectrum $S_\eta(\omega)\Delta\omega$ represents the variance of the n-th wave component:

$$a_n = \sqrt{2}\sqrt{S_\eta(\omega)\Delta\omega} \tag{2.3}$$

- The wave numbers k_n are obtained from the given frequency ω_n using the dispersion relation:

$$\omega^2 = k_n g \tanh(k_n h) \tag{2.4}$$

  being g the gravity acceleration and h the water depth. For deep water, this expression simplifies to

$$\omega^2 = k_n g \tag{2.5}$$

- The phase angles, φ_n, are discarded when the wave spectrum $S_\eta(\omega)$ is generated from the irregular wave history, so these are randomly selected on the range $0 \leq \varphi_n \leq 2\pi$ during the transformation to time record. The exact value of the phase angle correlated to the given frequency does not influence the statistics of the newly generated time history, therefore the new random set of φ_n values produce an instantaneously different but statistically and energetically equivalent time record (see Figure 2.5).

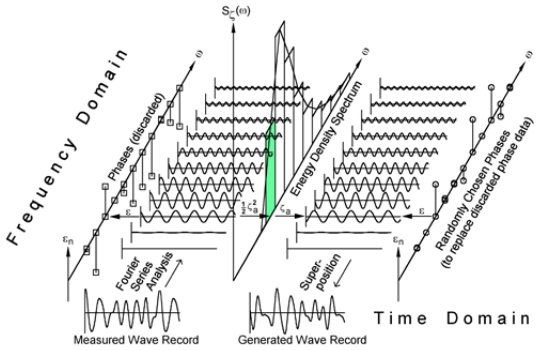


Figure 2.5: *Wave time record analysis (left) and regeneration (right). Taken from [7]*

2.2.2 Hydrodynamic forces

A floating body interacts with the fluid through different phenomena. To obtain an accurate modelling of the system dynamics, all must be taken into account. In this section, each source of loading is discussed.

Hydrostatic restoring force

As Archimede's principle says, "*any object, wholly or partially immersed in a fluid, is buoyed up by a force equal to the weight of the fluid displaced by the object*". This force is nothing but the vertical component of the hydrostatic force. In reality, the body receives both hydrostatic forces and moments:

$$\vec{F}_{hs} = \int_{S_0} p_{hs}\vec{n}dS \tag{2.6}$$

$$\vec{M}_{hs} = \int_{S_0} p_{hs}(\vec{r} \times \vec{n})dS \tag{2.7}$$

where $p_{hs} = -\rho g z$ is the hydrostatic pressure acting at depth z below the sea surface, $\vec{n}$ is the unit vector pointing outwards normally to the floater surface, $\vec{r}$ is the position vector of a point on the surface with respect to the center of gravity (COG) and S_0 is the floater's wetted surface. When the body is in its hydrostatic equilibrium position, hydrostatic forces and moments are compensated by the forces and moments exerted by the distributed gravity force.

When the body moves, *surge*, *sway* and *yaw* motions takes the body to a new equilibrium position, conversely, *roll* and *pitch* motions induce moments around the center of gravity, so, the behaviour of the floating body changes depending if the equilibrium position is stable, neutral or unstable. *Heave* motion does not perturb the equilibrium condition.

Considering an overall motion around a mean equilibrium floating position, buoyancy load is balanced by the weight of the body, and the remaining actions are a product of the displacements z, r_x and r_y:

- change in load caused by the change of the submerged volume of the body as it heaves, rolls and pitches, called *waterplane area effects*;
- change of the moment caused by movements of the COG and COB (Center of Buoyancy), the so called *moment arm effects*. Adopting the *small amplitude hypothesis*, the wetted surface and the displaced volume of fluid can be considered constants and equal to the ones at rest position, so, *waterplane area effects* can be neglected, while *moment arm effects* can be described by a linear relation.

Under this assumptions, the *hydrostatic restoring force* can be mathematically expressed through the following relation:

$$\vec{F}_{hsr} = -\bar{K}\vec{x} \tag{2.8}$$

Being $\vec{x}$ the vector containing the displacements of the floater COG relatively to the FRA and $\bar{K}$ the *hydrostatic stiffness matrix*:

$$\bar{K} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & K_{33} & K_{34} & K_{35} & 0 \\ 0 & 0 & K_{43} & K_{44} & K_{45} & K_{46} \\ 0 & 0 & K_{33} & K_{33} & K_{33} & K_{56} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Due to the small amplitude approximation, the coefficients contained inside $\bar{K}$ are all constant and moreover:

- the rotating blades and the nacelle have negligible masses and their centre of mass is very close to the z axis, so, being xz and yz symmetry planes, $K_{34} = K_{35} = K_{45} = 0$;
- in the small amplitude approximation the centre of buoyancy and the centre of gravity stay vertically aligned, so $K_{46} = K_{56} = 0$.

Froude-Krylov force

Is the resultant force caused by the unsteady pressure field generated by the undisturbed incident wave pressure field:

$$\vec{F}_{FK} = -\int_{S_0} p_I \vec{n} dS \tag{2.9}$$

Where p_I is the incoming wave pressure field.

Diffraction force

Is caused by waves created during the interaction between the incident wave and the body.

$$\vec{F}_D = -\int_{S_0} p_D \vec{n} dS \tag{2.10}$$

Where p_D is the unsteady pressure field associated with the diffracted wave field, and can be expressed using the diffraction forces *impulse response function matrix* $\bar{K}_D$:

$$\vec{F}_D = -\int_{S_0} \bar{K}_D(t-\tau)\eta(\tau)d\tau \tag{2.11}$$

Radiation force

Due to its movements, the floater generates radiated waves, having a pressure field p_R. The resultant force is

$$\vec{F}_R = -\int_{S_0} p_R \vec{n} dS \tag{2.12}$$

The radiation force can be written through the formulation of Cummins [8]:

$$\vec{F}_R(t) = -\bar{A}_\infty \ddot{\vec{x}} - \int_0^t \bar{K}_R(t-\tau)\dot{\vec{x}}(\tau)d\tau \tag{2.13}$$

Where $\bar{A}_\infty$ is a frequency-independent term, the constant and positive definite *added mass matrix*, that takes into account the water mass carried by the floater in its motion. The integral term is a convolution integral over speed $\dot{\vec{x}}(t)$ representing a source of damping. $\bar{K}_R$ is called *memory function matrix*.

Drag Force

Drag force takes into account the damping effect caused by fluid viscosity and flows detaching from the floater. Due to the fact that is a non-linear phenomenon, it can't be calculated through linear codes such as the one utilized in this work (see Section 2.2.3). For this reason, a model of quadratic drag force is implemented for each degree of freedom:

$$F_{drag,i} = -\frac{1}{2}\rho C_{d,i} A_i |\dot{x}_i| \dot{x}_i = -\beta_{d,i} |\dot{x}_i| \dot{x}_i \tag{2.14}$$

Where $C_{d,i}$ is the drag coefficient, A_i is the body cross sectional area and $\beta_{d,i} = \frac{1}{2}\rho C_{d,i} A_i$ is the condensed drag coefficient.

In the present work, for *surge* and *sway* motion a coefficient of drag in the form $C_{d,i}$ is used with values suggested by Ghosh, Islam and Ali [9] for an hexagonal cilinder, while, for *heave*, *roll*, *pitch* and *yaw* motions a condensed coefficient of drag is used performing an hydrodynamic scaling with values taken from the hydrodynamic modelling of the *DeepCwind* floating wind system for phase II of the Offshore Code Comparison Collaboration Continuation (OC4, see [10]). The scaling is based on the *Froude-Krylov scaling factor*:

$$\frac{\textit{floater wetted surface}}{\textit{reference floater wetted surface}} = \frac{A_w^{SF}}{A_w^{DC}} = \kappa = 1.342$$

$C_{d,x}$	0.95
$C_{d,y}$	0.80
$\beta_{d,z}$	$5.21 \cdot 10^6$
β_{d,r_x}	$4.97 \cdot 10^{10}$
β_{d,r_y}	$4.97 \cdot 10^{10}$
β_{d,r_z}	$5.48 \cdot 10^9$

Table 2.1: *Drag Coefficients*

where the superscripts "DC" and "SF" denote *DeepCWind* and *Seaflower*, respectively. The scaling is done, for the i-th degree of freedom according to

$$\beta_{d,i}^{SF} = \kappa \cdot \beta_{d,i}^{DC} \tag{2.15}$$

The adopted drag coefficients are shown in the Table 2.2.2.

2.2.3 Boundary Element Method analysis

The hydrodynamic analysis of the system is performed with a linear *Boundary Element Method* (BEM), one of the most extensively used models thanks to its numerical efficiency and accuracy. For this work, the software ANSYS Aqwa [11], running an implementation of the linear BEM through panel method (representation of the structure through diffraction panels) is used.

The base assumptions are, calling $\vec{v}(x, y, z, t)$ the flow velocity field and $\Phi(x, y, z, t)$ the velocity potential:

(*i*) irrotational flow $\hookrightarrow \vec{\nabla} \times \vec{v} = 0$

(*ii*) incompressible fluid, for which the continuity equation can be expressed as $\vec{\nabla} \cdot \vec{v} = 0$

Thanks to this base assumptions, the flow can be modelled as a potential flow, been guaranteed the existence of a potential function:

$$\vec{v}(x, y, z, t) = \vec{\nabla}\Phi(x, y, z, t) \tag{2.16}$$

under the above hypothesis, $\Phi(x, y, z, t)$ is nothing but the solution of the Laplace Equation

$$\nabla^2\Phi = 0 \tag{2.17}$$

Therefore, the problem is closed, and the boundary value problem (BVP) is:

$$\begin{cases} \nabla^2\Phi = 0 & P \in V & \text{Laplace Equation} \\ \dfrac{\partial\Phi}{\partial n} = \vec{v} \cdot \vec{n} & P \in S_0(t) & \text{slip condition on the body surface} \\ \dfrac{\partial\Phi}{\partial n} = 0 & P \in S_{bed} & \text{slip condition on the seabed surface} \\ \vec{v} \longrightarrow 0 & P \longrightarrow \infty & \text{unaffected far field velocity condition} \\ \dfrac{\partial\eta}{\partial t} + \vec{\nabla}\eta \cdot \vec{\nabla}\Phi & P \in S_{fs} & \text{kinematic free surface condition} \\ \dfrac{\partial\Phi}{\partial t} + g\eta + \dfrac{1}{2}\|\vec{\nabla}\Phi\|^2 = 0 & P \in S_{fs} & \text{dynamic free surface condition} \end{cases} \tag{2.18}$$

Aqwa makes additional assumptions in order to linearise the problem:

(iv) small wave steepness $\frac{H}{\lambda}$

(v) small amplitude motions

This means that the floater wetted surface $S_0(t)$ is considered constant with time and equal to the mean wetted surface S_0. Thus, the last two equations

of the previous system are replaced by a unique equation:

$$\begin{cases} \nabla^2\Phi = 0 & P \in V & \text{Laplace Equation} \\ \dfrac{\partial\Phi}{\partial n} = \vec{v}\cdot\vec{n} & P \in S_0 & \text{slip condition on the body surface} \\ \dfrac{\partial\Phi}{\partial n} = 0 & P \in S_{bed} & \text{slip condition on the seabed surface} \\ \vec{v} \longrightarrow 0 & P \longrightarrow \infty & \text{unaffected far field velocity condition} \\ \dfrac{\partial^2\Phi}{\partial t^2} + g\dfrac{\partial\Phi}{\partial z} = 0 & z = 0 & \text{dynamic free surface condition} \end{cases} \tag{2.19}$$

The linear BVP expressed by the system of equations 2.19 is solved by Aqwa using the Green's function in *frequency domain*, obtaining the potential flow function $\varphi(x, y, z)$. Due to the linearity of 2.19, the total flow potential can be seen as a superposition of different flow potentials:

$$\varphi(x, y, z)e^{-j\omega t} = \left(\varphi_I + \varphi_D + \sum_{i=1}^{6} \varphi_{R,i}x_i\right) e^{-j\omega t} \tag{2.20}$$

with: φ_I, φ_D, $\varphi_{R,i}$, referring to the incident, diffracted and radiated flow potentials.

The term $\varphi_{R,i}$ depends on the body motions, and thus the summation expresses the scalar product $\vec{\varphi}_R \cdot \vec{x}$.

Once the potentials are determined, the first order hydrodynamic pressure can be calculated using the linearised Bernoulli's equation:

$$p(x, y, z, t) = -\rho\frac{\partial\Phi(x, y, z, t)}{\partial t} = j\omega\rho\varphi(x, y, z)e^{-j\omega t} \tag{2.21}$$

By integrating the pressure distribution over the wetted surface of the floater, the different hydrodynamic forces and moments (see Section 2.2.2) can be obtained:

$$\vec{f}e^{-j\omega t} = -\int_{S_0} p(x, y, z, t)\vec{n}dS \tag{2.22}$$

From Equation 2.22, taking into account Equations 2.20 and 2.21, each component of the total force $\vec{f} = \vec{f_i} + \vec{f_D} + \bar{f}_r\vec{x} = \vec{f_I} + \vec{f_D} + \vec{f_R}$ is be calculated:

$$\begin{cases} \vec{f_I} = -j\omega\rho\displaystyle\int_S \varphi_I(x, y, z)\vec{n}dS & \text{Froude-Krylov force} \\ \vec{f_D} = -j\omega\rho\displaystyle\int_S \varphi_D(x, y, z)\vec{n}dS & \text{Diffraction force} \\ \bar{f}_R = -j\omega\rho\displaystyle\int_S \bar{\varphi}_R(x, y, z)\vec{n}dS & \text{Radiation force} \end{cases} \tag{2.23}$$

It's important to notice that $\bar{f}_R$ is a 6×6 matrix espressing the relation between the ith unit amplitude body rigid motion x_i (columns) and the hth generalized radiation force component $f_{Rh} = \bar{f}_{Rh} \cdot \vec{x}$ (rows).

The radiation force matrix $\bar{f_R}$ can be divided into real and imaginary part:

$$\bar{f_R} = -j\omega\rho \int_{S_0} [Re(\bar{\varphi}_R) + jIm(\bar{\varphi}_R)]\vec{n}dS = \omega^2 \bar{A}(\omega) + j\omega\bar{B}(\omega) \tag{2.24}$$

Where

$$\bar{A}(\omega) = \frac{\rho}{\omega} \int_{S_0} Im(\bar{\varphi}_R)\vec{n}dS \tag{2.25}$$

is the *Added Mass Matrix*, and

$$\bar{B}(\omega) = -\rho \int_{S_0} Re(\bar{\varphi}_R)\vec{n}dS \tag{2.26}$$

is the *Radiation-Damping Matrix*.

For this work, the coupling between DOFs consists of surge with pitch and sway with roll, because both xz and yz planes are simmetry planes:

$$\bar{A}(\omega) = \begin{pmatrix} A_{11}(\omega) & 0 & 0 & 0 & A_{15}(\omega) & 0 \\ 0 & A_{22}(\omega) & 0 & A_{24}(\omega) & 0 & 0 \\ 0 & 0 & A_{33}(\omega) & 0 & 0 & 0 \\ 0 & A_{42}(\omega) & 0 & A_{44}(\omega) & 0 & 0 \\ A_{51}(\omega) & 0 & 0 & 0 & A_{55}(\omega) & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{66}(\omega) \end{pmatrix}$$

$$\bar{B}(\omega) = \begin{pmatrix} B_{11}(\omega) & 0 & 0 & 0 & B_{15}(\omega) & 0 \\ 0 & B_{22}(\omega) & 0 & B_{24}(\omega) & 0 & 0 \\ 0 & 0 & B_{33}(\omega) & 0 & 0 & 0 \\ 0 & B_{42}(\omega) & 0 & B_{44}(\omega) & 0 & 0 \\ B_{51}(\omega) & 0 & 0 & 0 & B_{55}(\omega) & 0 \\ 0 & 0 & 0 & 0 & 0 & B_{66}(\omega) \end{pmatrix}$$

2.2.4 Hydrodynamic model in the *frequency domain*

For a floating body, the II law of motion is

$$\bar{M}\ddot{\vec{x}} = \vec{F}_{hsr} + \vec{F}_{FK} + \vec{F}_D + \vec{F}_R + \vec{F}_{drag} + \vec{F}_m + \vec{F}_{WT} \tag{2.27}$$

While the regular incoming wave, considered on the vertical axis z can be written as

$$\eta(t) = ae^{-j\omega t}$$

because the model is linear, also the excitation force has the same form: it is associated with the diffraction and Froude-Krylov forces:

$$\vec{F}_{exc}(t) = \vec{F}_{FK} + \vec{F}_D = a|\vec{f}(\omega)|e^{-j(\omega t + \angle \vec{f}(\omega))} \tag{2.28}$$

Where $\vec{f}(\omega)$ carries the Froude-Krylov coefficients: excitation force amplitude and phase with respect to that of the incident wave, respectively as real and complex part. So, the excitation force can be written in the frequency domain as

$$\vec{F}_{exc}(\omega) = a\vec{f}(\omega) \tag{2.29}$$

Assuming small amplitude motions around the mean equilibrium floating position, neglecting also non-linear terms, for what illustrated in the previous sections the equation of motion in the frequency domain is

$$[-\omega^2(\bar{M} + \bar{A}(\omega)) + j\omega\bar{B}(\omega) + \bar{K}]\vec{x}(\omega) = \vec{F}_{exc}(\omega) \tag{2.30}$$

The added mass matrix $\bar{A}(\omega)$, the radiation-damping matrix $\bar{B}(\omega)$, the hydrodynamic stiffness matrix $\bar{K}$ and the Froude-Krylov coefficient array $\vec{f}(\omega)$ are extrapolated from the analysis performed with ANSYS Aqwa, for a given direction of the incident wave.

2.2.5 Hydrodynamic model in the *time domain*

The representation of the equation of motion for a floating body in time domain is due to Cummins [8]:

$$(\bar{M} + \bar{A}_\infty)\ddot{\vec{x}} + \int_0^t \bar{K}_R(t-\tau)\dot{\vec{x}}(\tau)d\tau + \bar{K}\vec{x}(t) = \vec{F}_{exc}(t) \tag{2.31}$$

There's also a relationship, discovered by Ogilvie [12], between time and frequency representations, obtained using the Fourier Transform on Equation 2.30:

$$\bar{A}(\omega) = \bar{A}_\infty - \frac{1}{\omega}\int_0^\infty \bar{K}_R(t)\sin(\omega t)dt \tag{2.32}$$

$$\bar{B}(\omega) = \int_0^\infty \bar{K}_R(t)\cos(\omega t)dt \tag{2.33}$$

It can be shown that the added mass matrix $\bar{A}(\omega)$ can be obtained as

$$\bar{A}_\infty = \lim_{\omega\to\infty}\bar{A}(\omega)$$

While the retardation function $\bar{K}_R$ can be written respectively in time and frequency domain as

$$\bar{K}_R(t) = \frac{2}{\pi}\int_0^\infty \bar{B}(\omega)\cos(\omega t) \tag{2.34}$$

$$\bar{K}_R(j\omega) = \int_0^\infty \bar{K}_R(t)e^{-j\omega t}d\omega = \bar{B}(\omega) + j\omega[\bar{A}(\omega) - \bar{A}_\infty] \tag{2.35}$$

The implementation of Equation 2.31 is highly time and memory-demanding because of the presence of the convolution integral associated with the radiation damping. For this reason, the approximation proposed by Perez [14][15] and based on a linear and time invariant *State Space Model* is used, solving instead of the convolution integral a set of linear ordinary differential equations for each degree of freedom:

$$\vec{\mu}(t) = \int_0^t \bar{K}_R(t-\tau)x_i(\tau)d\tau \approx \begin{cases}\dot{\vec{u}}(t) = \bar{A}_{ss}\vec{u}(t) + \bar{B}_{ss}\dot{\vec{x}}(t)\\ \vec{\mu}(t) = \bar{C}_{ss}\vec{u}(t)\end{cases} \tag{2.36}$$

Where $\vec{x}(t)$ is the input vector, $\vec{\mu}(t)$ the output vector, and $\vec{u}(t)$ the state space vector of the state space model. $\bar{A}_{ss}$, $\bar{B}_{ss}$ and $\bar{C}_{ss}$ are to be estimated through model identification.

2.2.6 Frequency Domain Analysis

To obtain all the parameters needed to run the hydrodynamic time-domain model of the system, a simulation using the software ANSYS Aqwa is started. The software needs a CAD model of the floating body, realized using the commercial CAD/CAE software SolidWorks by Dassault Systèmes and the geometrical data, provided by Fincantieri and representative of the whole structure (wind turbine + floating platform + moorings). ANSYS Aqwa *"provides a toolset for investigating the effects of environmental loads on floating and fixed offshore and marine structures, [...]floating production and offloading systems, spars, semi-submersibles, renewable energy systems, and ships. [...][Aqwa] can perform frequency domain statistical analysis and real-time motion of a floating body or bodies while operating in regular or irregular waves, [...] in which nonlinear Froude-Krylov and hydrostatic forces are estimated under instantaneous incident wave surface; [...] Wind and current loading can also be applied to the bodies, as well as external forces"[11].*

Figure 2.6: *CAD model used to perform the analysis*

In this case is employed the "Hydrodynamic Diffraction" tool. The first step of the simulation is the geometry import. Once done it, the solid is repositioned coherently with the FRA reference frame, the xy plane being coincident with the still water plane. The solid must be reduced to boundary surfaces with zero thickness and normal vector pointing outwards, splitted at the water line and all structures originated grouped into a unique part. After, a point mass is added in the position of the center of gravity of the whole system, defining for it the mass and rotational inertia properties in terms of radii of gyration or mass moments of inertia. The environmental properties are defined, modelling the sea as a $1000m \times 1000m \times 1000m$ cube, with water density $\rho_w = 1025kg/m^3$. The gravity acceleration is set to $9.80665m/s^2$.

The analysis is performed for a range of incoming wave directions spanning

the round angle from -180° (x direction) to 180° (x direction) with intervals of 10°, to obtain a sufficient precision in the interpolation of force values in the *waves block* (see Section 2.2.7). The regular wave period range is from 2.5s to 40s with intermediate values every 0.5s, so to cover the typical spectrum of periods of oceanic waves. The highest period (lowest frequency) is limited by the water depth chosen and the lowest period (highest frequency) by the mesh size [11]. This limitation is mathematically formulated as:

$$L_{max} \leq \frac{1}{6}\lambda_w = \frac{2\pi}{6k}$$

where L_{max} is the maximum panel size and λ_w the wavelength. The above formulation can be rearranged, considering that we are dealing with regular waves and deep water ($d/\lambda_w > 0.5$, meaning that $k = \omega^2/g$, from the dispersion relation 2.4)

$$L_{max} \leq \frac{2\pi g}{6\omega^2} \rightarrow T \geq \sqrt{\frac{12\pi L_{max}}{g}}$$

For the present work, a mesh size of 1.3m is chosen, with a *defeaturing tolerance* of 0.75m. For the convergence analysis, see [13].

Once the analysis is accomplished, all parameters needed to run the simulation, including the hydrostatic stiffness matrix $\bar{K}$, the Froude-Krylov coefficients array $\vec{f}(\omega)$, the added mass matrix $\bar{A}(\omega)$ and the radiation-damping matrix $\bar{B}(\omega)$ are extrapolated.

2.2.7 Implementation of diffraction and Froude-Krylov Forces in Simulink

Allowing a variable incoming wave direction creates an issue: when the wave direction changes, also the wave-generated forces change. For this reason, there's the need to discretize the round angle computing the wave forces for each direction. In this work, the round angle is discretized in 36 different directions, with intervals of 10°, and, for each timestep t_i, forces for all the directions and the degrees of freedom are computed: if m is the number of time intervals of the record, and m is the number of directions analysed, then the wave forces will be stored inside an $m \times n \times 6$ tridimensional matrix in the following way

$$\bar{F}_k = \begin{pmatrix} F(t_1,\theta_1) & F(t_1,\theta_2) & \cdots & F(t_1,\theta_j) & \cdots & F(t_1,\theta_n) \\ F(t_2,\theta_1) & F(t_2,\theta_2) & \cdots & F(t_2,\theta_j) & \cdots & F(t_2,\theta_n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ F(t_i,\theta_1) & F(t_i,\theta_2) & \cdots & F(t_i,\theta_j) & \cdots & F(t_i,\theta_n) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ F(t_m,\theta_1) & F(t_m,\theta_2) & \cdots & F(t_m,\theta_j) & \cdots & F(t_m,\theta_n) \end{pmatrix} \tag{2.37}$$

Another critical point is that the wave force record is generated using a different time step with respect to the one used by the simulation, generally. For an

example, if the simulation is at time t_α, there's the possibility that at that time there is no waveforce record available, being the same reasoning valid also for the incoming wave direction. For this reason, the various t_i and θ_j in which the record is discretized are stored inside two arrays:

$$\vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_i \\ \vdots \\ t_m \end{pmatrix} \tag{2.38}$$

$$\vec{\theta} = \begin{pmatrix} \theta 1 \\ \theta_2 \\ \vdots \\ \theta_i \\ \vdots \\ \theta_n \end{pmatrix} \tag{2.39}$$

When the simulation goes on, the block implementing the wave forces receives the simulation time τ and compares it with the elements stored in $\vec{t}$, obtaining the index k_t of the interval (t_i, t_{i+1}) in which τ is and the normalized position f_t of it inside the interval. The same happens with the wave direction imposed, γ, obtaining the index k_θ and the normalized position f_θ. Once the indexes and the normalized positions are obtained, the model linearly interpolates between the values stored inside the matrix $\bar{F}$, obtaining the value of the wave force.

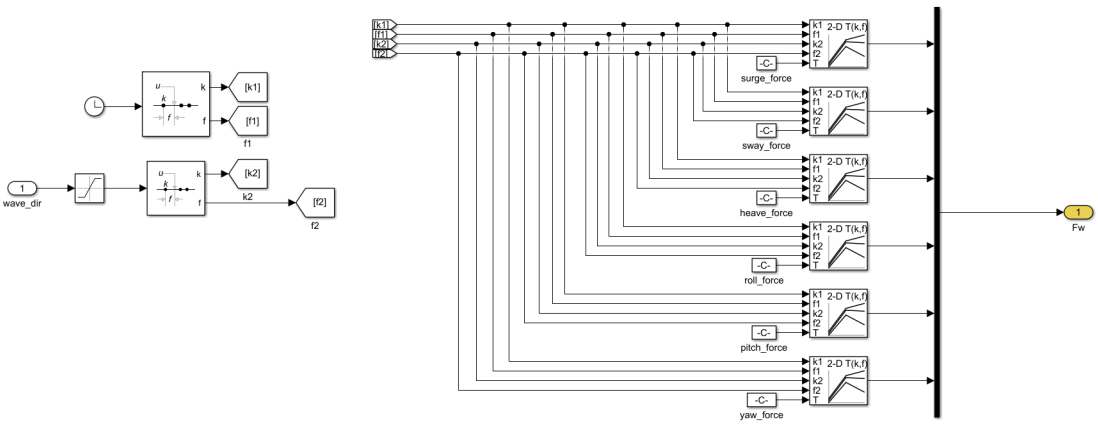


Figure 2.7: *Diffraction and Froude-Krylov Forces implementation*

2.2.8 State-Space model and Stability Analysis

The State-Space matrices are obtained starting from added mass matrix $\bar{A}(\omega)$ and radiation-damping matrix $\bar{B}(\omega)$, using the Matlab© tool provided by [16]. The approximation problem expressed by Equation 2.36 can be rearranged as

$$\bar{K}(j\omega) \approx \hat{K}(s) = \bar{C}_{ss}(j\omega\bar{I} - \bar{A}_{ss})^{-1}\bar{B}_{ss} \tag{2.40}$$

where $\hat{K}(j\omega)$ is matrix rational transfer function with entries

$$\hat{K}_{ij}(s) = \frac{P_{ij}(s)}{Q_{ij}(s)} = \frac{p_r s^r + p_{r-1}s^{r-1} + ... + p_0}{s^n + q_{n-1}s^{n-1} + ... + q_0} \tag{2.41}$$

Figure 2.8: Linear Time Invariant State Space Model proposed by [14]

this means that, by estimating the transfer functions $\hat{K}_{ik}(s)$ (see [16]) one can obtain the matrices of the State Space model, that may not necessarily be stable, because stability is not enforced as a constraint. This means that, before feeding the State Space matrix into the time domain model, a stability verification must be done.

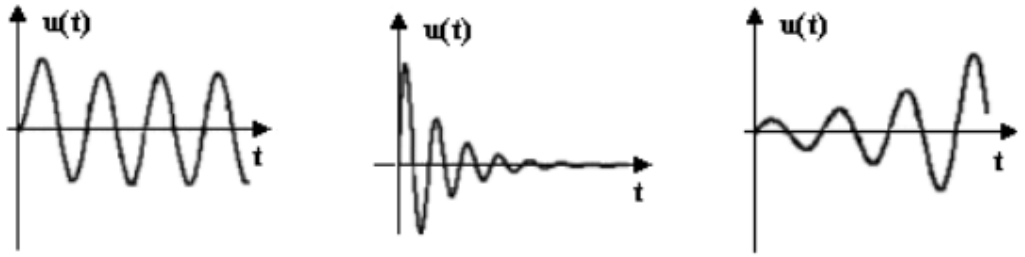
Recalling that a Linear Time Invariant system is

- *Simply stable* if it responds to a limited input with a limited output;
- *Asimptotically stable* if it responds with an output that tends to zero as $t \to \infty$;
- *Unstable* if it responds with an unlimited output to a limited input;

and that stability of a system doesn't depend on the input signal, but only on its transfer function $f(s)$, is possible to obtain informations about the stability of a system knowing poles (roots of the denominator polynomial) and zeroes (roots of the numerator polynomial) of $f(s)$ using the *General Stability Criterion*, that relates the position of zero and poles of a certain transfer function on the Gaussian plane with its dynamical behaviour. It says that a system is:

(*i*) *simply stable* if and only if all the poles of its transfer function have negative real part;

(*ii*) *asimptotically stable* if and only if all the poles of its transfer function have negative or null real part, at least one has null real part, and all poles with null real part are simple;



a) b) c)

Figure 2.9: *a) Simply Stable system b) Asimptotycally stable system c) Unstable system*

(*iii*) *unstable* if and only if at least one pole of its transfer function has positive real part or its multiple;

In the case of the LTI SSM (Linear Time-Invariant State Space Model) described in this work, the transfer function has the form of the matrix $\hat{K}(s)$ of equation 2.40, and is a function of the State Space Matrices $\bar{A}_{ss}, \bar{B}_{ss}, \bar{C}_{ss}$.

The analysis is done through the Matlab© tool `pzplot`, obtaining 10 pole-zero maps, one for each coupling, that can be found in Appendix A.

2.3 Moorings

2.3.1 Elastic Model

In this work, an elastic model of the mooring lines is implemented, with the following hypothesis:

(i) dynamic actions are neglected;

(ii) the only effect of mass is considered to be the weight force;

(iii) all sorts of damping sources coming from the mooring lines are neglected, including hydrodynamic resistance, friction with the seabed, ropes inner friction and friction at the connection points;

(iv) a purely elastic behaviour of all elements composing the mooring lines is considered;

(v) the chain segments are modelled as continous bodies, and the lines are considered to be directly connected to the platform and to the anchors, with no intermediate parts. Fairleads are dimensionless, and located at each vertex of the hexagonal base of the floater;

(vi) the anchors at the seabed are considered as fixed.

2.3.2 Moorings Dynamics modelling

Modelling the mooring lines as linear springs makes necessary to calculate, at every instant, the length of the polyester rope and the relative distance between the anchor and the fairlead for each line.

Waves induces motions of the floater, and the connection points C_i move in the space, determining a change of the length of the lines and in the tension. So, is necessary to compute the position of the connection points in the FRA reference frame at every instant.

The generic movement of the floater is a roto-translation (see Figure 2.10) being $\vec{t} = (\Delta x, \Delta y, \Delta z)$ the translations vector and $\vec{r} = (\Delta r_x, \Delta r_y, \Delta r_z)$ the rotations vector. The motion of the connection point C_i is thus the composition of a translation of the center of mass and a rotation of the whole structure around G, the center of gravity. This results in the change in length of the mooring line from $\|\vec{OC}\|$ to $\|\vec{OC'}\|$. At rest, the center of mass G coordinates in the FRA reference frame are:

$$\vec{x}_G = (0, 0, z_G)$$

When the body translates, the center of mass will occupy a new position in the FRA:

$$\vec{x}'_G = (\Delta x, \Delta y, z_G + \Delta z) = \vec{t}$$

Consequently, the new coordinates of the connection point C_i, considering only the translation contribution will be:

$$\vec{x}'_{C_i,t} = \vec{x}'_G = \vec{t}$$

The rotation of C_i around the center of mass G is given by the composition of the three left-handed rotations around the three axes of the FRA:

$$\bar{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(r_x) & -\sin(r_x) \\ 0 & \sin(r_x) & \cos(r_x) \end{pmatrix} \tag{2.42}$$

$$\bar{R}_y = \begin{pmatrix} \cos(r_y) & 0 & \sin(r_y) \\ 0 & 1 & 0 \\ 0 & \sin(r_x) & \cos(r_x) \end{pmatrix} \tag{2.43}$$

$$\bar{R}_z = \begin{pmatrix} \cos(r_z) & -\sin(r_z) & 0 \\ \sin(r_z) & \cos(r_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.44}$$

Being the complete rotation represented by their matrix multiplication

$$\bar{R}_{xyz} = \bar{R}_x \bar{R}_y \bar{R}_z \tag{2.45}$$

By multiplying this matrix with the position vector $\vec{x}_{C,LSA}$ of the connection point in the LSA reference frame, the rotation contribution to the global movement is

$$\vec{x}'_{C,r} = \bar{R}_{xyz} \vec{x}_{C,LSA} \tag{2.46}$$

Then, the position vector of the connection point after a generic motion of the floater is

$$\vec{x}'_C = \vec{x}'_{C,r} + \vec{x}'_{C,t} \tag{2.47}$$

Once the position vector $\vec{x}'_C$ is known, the forces and moments exerted by the mooring system can be calculated. The tension $\vec{T}_c$ at the connection point lies on the straight line passing through the anchor point and the connection point C, and its magnitude can be described by a piecewise function:

$$T_c = \begin{cases} T_0 + k(L' - L) & L' > L \\ 0 & L' \leq L \end{cases} \tag{2.48}$$

Where k is the line stiffness, L its unstretched length, $L' = \|\vec{OC'}\|$ the instantaneous stretched length and T_0 the pre-tension value. The tension vector is thus:

$$\vec{T}_C = T_c \frac{\vec{OC'}}{\|\vec{OC'}\|} \tag{2.49}$$

This force produces also moments on the structure, expressed by

$$\vec{M}_c = \vec{x}'_{C,r} \times \vec{T}_C \tag{2.50}$$

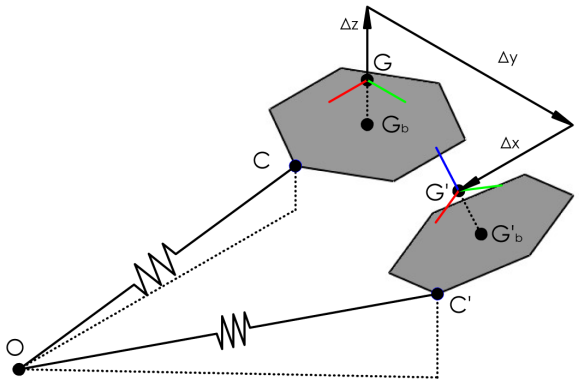


Figure 2.10: *Generic motion of the floater and change in mooring line length. Taken from [13]*

2.4 Wind Turbine

The wind turbine is simulated externally from the *floater module*, using the software *QBlade*, an open source software developed by the Hermann Föttinger Institute of the TU Berlin[1]. It implements a *Non Linear Lifting Line Free Vortex Wake Method*, belonging to the big family of the *Vortex Methods*, that, in terms of accurate modelling of the physics and computational cost are on the mid-way between the *Blade Element Momentum* (BEM) methods and the *Computational Fluid Dynamics*. One large advantage of the vortex methods, compared to the BEM methods, is that due to the sound modelling of the macroscopic physics, only very few empirical assumptions related to the microscopic fluid dynamics, where boundary layers effects play an important role, such as dynamic stall or stall delay need to be added [17]. It's important to remark that this method is not only able to provide results concerning the rotor performances and blade loads, but also to provide a solution for the unsteady velocity field around the rotor, and the wake.

In this section, the mathematical model on which the method is based is exposed, however, some basic concepts of aerodynamics are recalled.

2.4.1 Aerofoils

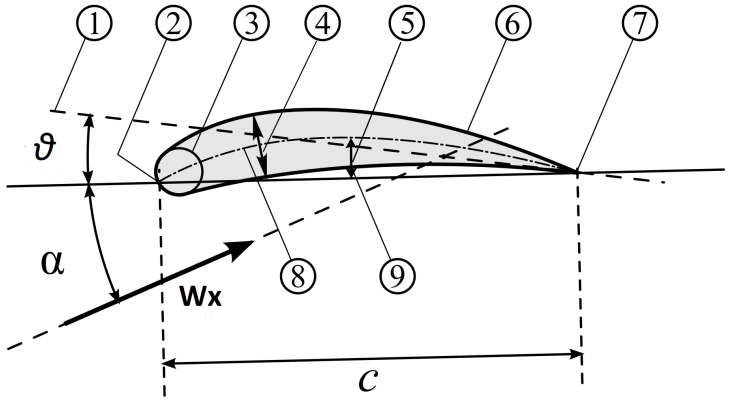


Figure 2.11: *Geometrical characteristics of an aerofoil.*

The cross section of a wind turbine blade is an aerofoil. In Figure 2.11 can be identified:

1. *rotor plane*;
2. *leading edge*;
3. *nose circle*;
4. *section of maximum thickness*
5. *camber*, the maximum distance between the chord and the camber line;

[1] `http://www.q-blade.org/`

6. *upper surface*;

7. *trailing edge*;

8. *camber line*, the line joining the trailing and the leading edges being the upper and the lower surfaces of the aerofoil always at the same distance from it;

9. *lower surface*

The *chord*, c, is defined as the line joining the leading and the trailing edges, the *angle of attack* α as the angle between the relative wind speed (in Figure 2.11 identified by W_x), while the *pitch angle* is the angle between the rotor plane and the chord line. The forces acting on the aerofoil are usually considered as applied at the quarter chord, as can be seen in Figure 2.12: the resultant force F_{ris} can be decomposed into two components, *lift* and *drag*, that are respectively the force acting perpendicularly to the relative wind speed and the force acting parallel to it, plus a moment, M.

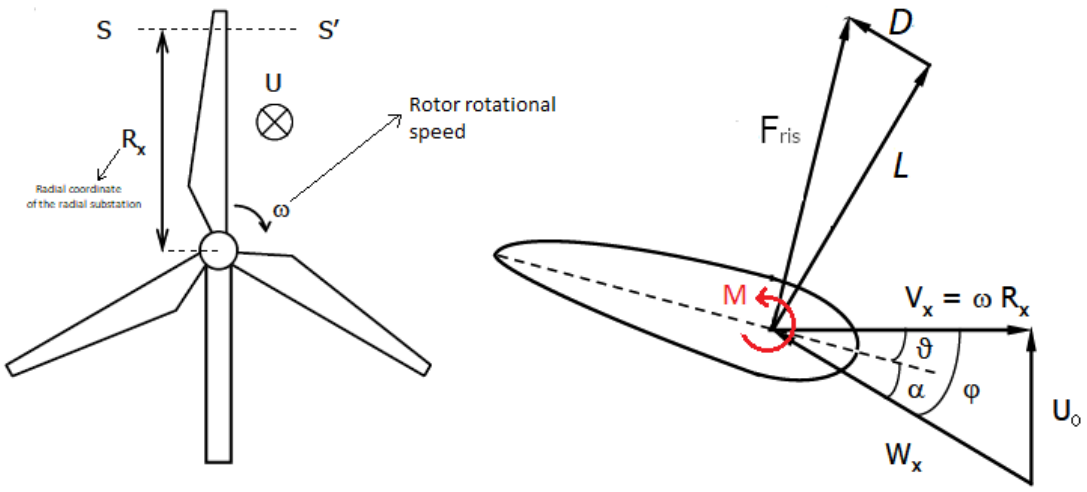


Figure 2.12: *Generic aerofoil and forces acting on it.*

Then, three non-dimensional quantities, respectively the lift, drag and moment coefficient can be introduced:

$$C_L = \frac{L}{0.5\rho U_0^2 c} \tag{2.51}$$

$$C_D = \frac{D}{0.5\rho U_0^2 c} \tag{2.52}$$

$$C_M = \frac{M}{0.5\rho U_0^2 c^2} \tag{2.53}$$

2.4.2 Vortex Dynamics

The governing set of equations for an incompressible fluid flowing are the Navier-Stokes equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla} \cdot \vec{u} = -\frac{1}{\rho}\nabla p + \mu \nabla^2 \vec{u} \tag{2.54}$$

where $\vec{u} = (u, v, w)$ is the velocity vector, ρ the fluid density and μ the fluid viscosity. That, together with the continuity equation for an incompressible flow

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = \nabla \vec{u} = 0 \tag{2.55}$$

Permit, in principle, to obtain the fluid velocity and the pressure field. However, these equation cannot be solved analytically, and, when solved numerically, the problem of very high computational cost arises.

However, flow fields can be described using different mathematical tools, presented in this subsection. When an incompressible fluid element moves in a flow field, it experiences a *translation*, and a *distorsion*, this last one being composed by *rotation* and *shear*, in turn caused by the viscosity of the fluid. The instantaneous rate of rotation of a fluid element is given, in the three dimensions, by the *vorticity* vector, defined as

$$\vec{\Omega} = (\xi, \eta, \varsigma) = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) \tag{2.56}$$

Vorticity is solenoidal:

$$\nabla \cdot \vec{\Omega} = \frac{\partial \xi}{\partial x} + \frac{\partial \eta}{\partial y} + \frac{\partial \varsigma}{\partial z} = 0 \tag{2.57}$$

Considering an irrotational flow, the vorticity $\vec{\Omega}$ must be equal to zero everywhere in the flow domain and the velocity vector must satisfy $\nabla \times \vec{u} = 0$, implying the existence of a potential function

$$\vec{u} = (u, v, w) = \left(\frac{\partial \varphi}{\partial x}, \frac{\partial \varphi}{\partial y}, \frac{\partial \varphi}{\partial z} \right) \tag{2.58}$$

Using this property, together with the irrotational flow assumption and the continuity equation for a tridimensional flow expressed by Equation 2.55, the linear *Laplace's Equation* can be obtained:

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2} = \nabla^2 \varphi = 0 \tag{2.59}$$

While Equation 2.56 can be inverted to give the velocity field as an integral over the vorticity field [18]

$$\vec{u} = \vec{u}_\Omega + \nabla \varphi \tag{2.60}$$

Where $\vec{u}_\Omega$ is given by the integral over the flow domain

$$\vec{u}_\Omega = \frac{1}{4\pi}\int_V \frac{\vec{\Omega}\times\vec{r}}{r^3} \tag{2.61}$$

and φ can be determined solving the potential flow problem related to the flow around a body, and is called the *disturbance potential*. It is given by two contributions:

$$\varphi = \varphi_\infty + \varphi_\sigma \tag{2.62}$$

Where φ_∞ is the contribution related to the undisturbed fluid flowing around the body, and φ_σ is related to the distribution of point sources, a point at which the fluid is appearing at a uniform rate $\sigma[m^3/s]$ placed around the body to model the flow. This description of the flow field could result more convenient, because if the fluid is barotropic and external forces are conservative, then the vorticity satisfies the Helmholtz theorems:

1. The strength of a vortex filament is constant along its length, which must be either closed or end at the boundary of the fluid

$$\int_S \vec{\Omega}\cdot\vec{n} = \oint_C \vec{u}\cdot\vec{t}ds = \Gamma \tag{2.63}$$

   The quantity Γ is called the *circulation*, and is nothing but the flux of the vorticity vector through a surface of area A. It is called also *vorticity strength*.

2. The vortex strength is constant in time

$$\frac{d}{dt}\int_S \vec{\Omega}\cdot\vec{n}dS = 0 \tag{2.64}$$

2.4.3 Lifting Line Free Vortex Wake Method

The mathematical model on which *QBlade* is based, described in this subsection, is basically the one provided by van Garrel [19] and is here summarized. The flow field around the body is represented through a distribution of sources of strength σ, required to satisfy flow tangency on the surface (thickness effect) and vortices $\vec{\omega}$, that are used to satisfy the *Kutta Condition*: the flow must leave the aerofoil smoothly from the trailing edge. The fluid velocity is then given by the vectorial summation of the undisturbed velocity and the velocity "induced" by the distribution of vortices and sources

$$\vec{u} = \vec{u}_\infty + \vec{u}_\sigma + \vec{u}_\Omega \tag{2.65}$$

where

$$\vec{u}_\sigma = \frac{1}{4\pi}\int_V \frac{\sigma\vec{r}}{r^3}dV \tag{2.66}$$

is the velocity induced by the point sources in the flow domain, and

$$\vec{u}_\Omega = \frac{1}{4\pi}\int_V \frac{\vec{\Omega}\times\vec{r}}{r^3} \tag{2.67}$$

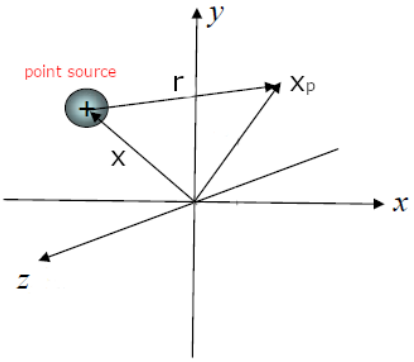


Figure 2.13: *Point Source.*

is the velocity induced by the vortices at the evaluation point $\vec{x}_p$ (see Figure 2.13), being $\vec{r} = \vec{x}_p - \vec{x}$ and r its norm.

All viscosity effects are taken into account using relationships between the angle of attack and lift, drag and pitching moment coefficients and the local flow velocity is supposed to be much smaller than the speed of sound: for this reason, the incompressible flow assumption holds. Also thickness and displacements effects are neglected, so sources have not to be modelled.

The momentum equation for an inviscid fluid can be written as

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla} \cdot \vec{u} = -\frac{1}{\rho}\nabla p + \frac{\vec{f}}{\rho} \tag{2.68}$$

Where $\vec{f}$ is the total external applied force in $[N/m^3]$. Using the identity

$$\vec{u} \cdot \vec{\nabla} \cdot \vec{u} = (\nabla \times \vec{u}) \times \vec{u} + \frac{1}{2}\nabla \vec{u}^2 = (\vec{\Omega} \times \vec{u}) + \frac{1}{2}\nabla \vec{u}^2 \tag{2.69}$$

its expression becomes

$$\frac{\partial \vec{u}}{\partial t} + (\vec{\Omega} \times \vec{u}) = -\nabla\left(\frac{p}{\rho} + \frac{\vec{u}^2}{2}\right) + \frac{\vec{f}}{\rho} \tag{2.70}$$

Taking the integral over the volume defined by the flow domain and applying the Gauss Theorem:

$$\int_V \frac{\partial \vec{u}}{\partial t} dV + \int_V (\vec{\Omega} \times \vec{u}) dV = -\int_S \left(\frac{p}{\rho} + \frac{\vec{u}^2}{2}\right) \cdot \vec{n} dS + \int_V \frac{\vec{f}}{\rho} dV \tag{2.71}$$

This equation expresses the equilibrium of the total pressure force acting over the surface of V together with the external force and the *vortex force*, an equivalent body force. If no external force acts *outside* V, and the total pressure $p_t = \dfrac{p}{\rho} + \dfrac{\vec{u}^2}{2}$ is constant over the boundary (actually true if the boundary is a stream surface) then the external force acting on the flow domain can be calculated as

$$\vec{F} = \int_V \rho(\vec{u} \times \vec{\Omega}) dV \tag{2.72}$$

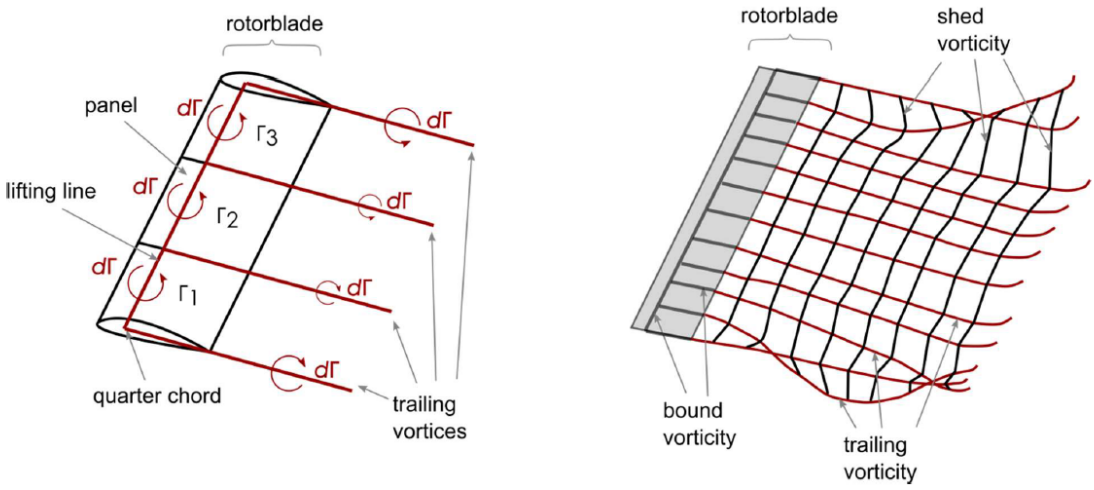


Figure 2.14: *Blade and wake modelling in the lifting line free vortex wake algorithm. Taken from [17]*

The vorticity $\vec{\Omega}$ inside the flow domain, surrounding the aerofoil can be concentrated into *vortex lines*, acting on the quarter chord of the aerofoil section. Considering then a vortex line element with oriented length $\vec{dl}$,Equation 2.72 can be rewritten as, considering the positive vortex line direction the one of the lumped volume vorticity distribution

$$d\vec{L} = \rho(\vec{u} \times \vec{\Gamma})dl = \rho\Gamma(\vec{u} \times d\vec{l}) \tag{2.73}$$

while the velocity "induced" by the distribution of vortices is expressed by Equation 2.67, and can be rearranged, using the Biot-Savart law, as

$$\vec{u}_\Gamma(\vec{x}_p) = -\frac{1}{4\pi}\int \Gamma \frac{\vec{r} \times d\vec{l}}{r^3} \tag{2.74}$$

for a straight line element, with constant circulation Γ, Equation 2.74 has an analytical expression

$$\vec{u}_\Gamma(\vec{x}_p) = \frac{\Gamma}{4\pi}\frac{(r_1 + r_2)(\vec{r}_1 \times \vec{r}_2)}{r_1 r_2(r_1 r_2 + \vec{r}_1 \cdot \vec{r}_2)} \tag{2.75}$$

The problem of is that if the evaluation point approaches the vortex line, Equation behaves singularly. For this reason, a new parameter, the *cut off radius*, δ, is introduced and Equation 2.4.3 is rewritten as

$$\vec{u}_\Gamma(\vec{x}_p) = \frac{\Gamma}{4\pi}\frac{(r_1 + r_2)(\vec{r}_1 \times \vec{r}_2)}{r_1 r_2(r_1 r_2 + \vec{r}_1 \cdot \vec{r}_2) + (\delta l_0)^2} \tag{2.76}$$

Where l_0 is the length of the vortex line element. The cut-off radius has a strong influence in the proximity of the vortex line element.

Vortex Wake

The blade geometry is modelled using one or more strips carrying a vortex ring whose bound vortex are located at the quarter chord, and at the trailing edge.

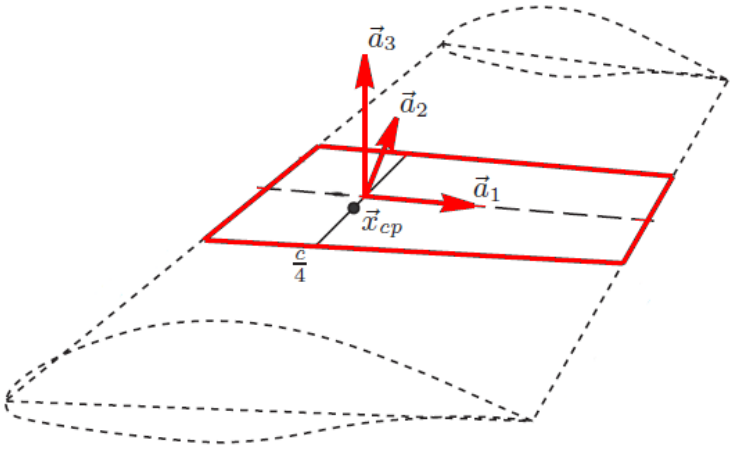


Figure 2.15: *Blade Strip.*

This vorticity, which vortex strength Γ has to be determined, is convected downstream as time goes by, creating the wake: at each timestep Δt vortex rings are shed from the trailing edge, joined with the older vortex rings and replaced. This convection mechanism is determined by the onset wind velocity and by the induced vorticity velocity in two steps:

$$\Delta\vec{x} = \vec{u}_{wind}\Delta t \tag{2.77}$$

$$\Delta\vec{x} = \vec{u}_{\Gamma}\Delta t \tag{2.78}$$

Vortex-Line Strength Computation

The vortex line strength, Γ, is determined matching the lift force as expressed by Equation 2.73 with the lift force associated with the local flow direction. This can be obtained from the aerodynamic tables, relating the local angle of attack α to the lift coefficient C_L:

$$dL = \frac{1}{2}C_L(\alpha)\rho U^2 dA \tag{2.79}$$

Being U the *strip* (spanwise element of blade) onset velocity magnitude and dA the strip area. But the vortex line problem is non-linear: the lift for each strip depends, as stated by the above equation, on the local flow velocity direction and magnitude. This lift in turn influences the strip vortex, as stated by Equation 2.73. The strip vortex influences the flow field according to 2.76, and therefore the lift of each strip. Matching this two expressions is done at the cross-section plane defined by the unit vectors in the chordwise and normal direction, respectively $\vec{a}_1$ and $\vec{a}_3$ (see Figure 2.15). At the quarter cord, as stated by Equation 2.76, we have:

$$dL_{\Gamma} = \rho\Gamma\sqrt{\left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_1\right)^2 + \left((\vec{u}_{cp}\times d\vec{l})\cdot\vec{a}_3\right)^2} \tag{2.80}$$

being $\vec{u}_{cp}$ the total onset velocity at the control point $\vec{x}_{cp}$, given by the wind, motion of the rotor and vorticity contributions of *all* the lifting line and wake elements

$$\vec{u}_c p = \vec{u}_{wind} + \vec{u}_{motion} + \vec{u}_\Gamma \tag{2.81}$$

The wind offset velocity is considered to be a function of time, known at each timestep. The motion-related speed is computed using the current and the previous position of the control point:

$$\vec{u}_{motion} = -\frac{\vec{x}_{cp,n} - \vec{x}_{cp,n-1}}{\Delta t} \tag{2.82}$$

where $\vec{x}_{cp,n}$ is the position of the control point at the n-th timestep and $\vec{x}_{cp,n-1}$ its position at the *(n-1)*-th timestep.

Then the lift force is calculated as

$$dL_\alpha = \frac{1}{2} C_L(\alpha_{cp}) \rho \Big((\vec{u}_{cp} \cdot \vec{a}_1)^2 + (\vec{u}_{cp} \cdot \vec{a}_3)^2 \Big) dA \tag{2.83}$$

being α_{cp} the angle of attack at the control point, defined as

$$\alpha_{cp} = \arctan \frac{\vec{u}_{cp} \cdot \vec{a}_1}{\vec{u}_{cp} \cdot \vec{a}_3} \tag{2.84}$$

Equating the two expression of the lift force, the vortex strength Γ can be obtained as

$$\Gamma = \frac{C_L(\alpha_{cp})}{2} \frac{(\vec{u}_{cp} \cdot \vec{a}_1)^2 + (\vec{u}_{cp} \cdot \vec{a}_3)^2 dA}{\sqrt{\Big((\vec{u}_{cp} \times d\vec{l}) \cdot \vec{a}_1 \Big)^2 + \Big((\vec{u}_{cp} \times d\vec{l}) \cdot \vec{a}_3 \Big)^2}} \tag{2.85}$$

The lift of each strip depends by the local flow velocity direction and magnitude, as stated by Equation Equation 2.83. This lift in turn fixes the strip vortex strength through Equation 2.73. The strip vortex ring with this strength acts on the complete flow field according to Equation 2.76, influencing the lift of each strip. So, the solution of Equation 2.85 requires an iterative procedure described in Figure 2.16.

Himmelskamp effect

The effects of rotation on the rotating blade was introduced by Himmelskamp [20]: for a given angle of attack, this tri-dimensional effect enhances the coefficient of lift compared to the one given by the two-dimensional aerofoil data, and stall is delayed. The Coriolis force, acting in the chordwise direction, causes a favourable pressure gradient that delays the flow separation, while the centrifugal force, acting in the spanwise directions, causes the boundary layer thinning. Because of the global increase of the lift coefficient, more pronounced near the root of the blade (regions of higher c/r ratio), the power output of the wind turbine will be enhanced. In *QBlade* a correction to take into account this effect is implemented:

$$C_{L,3D} = C_{L,2D} + \frac{3.1\lambda^2}{1+\lambda^2} g \Big(\frac{c}{r} \Big)^2 \Big(\frac{dC_l}{d\alpha} (\alpha - \alpha_0) - C_{L,2D} \Big) \tag{2.86}$$

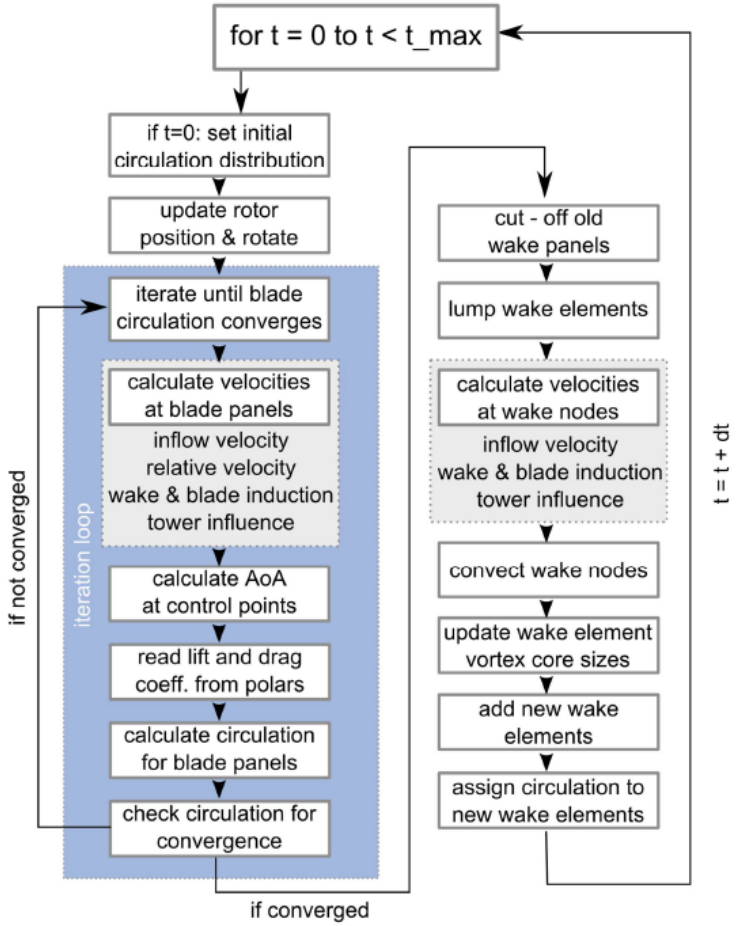


Figure 2.16: *Flowchart of the implemented lifting line algorithm*

being λ the *tip speed ratio*, g the gravity acceleration, $C_{L,3D}$ the corrected coefficient of lift, $C_{L,2D}$ the lift coefficient given by the aerofoil data, α the angle of attack for rotating blades and α_0 the static angle of attack.

2.4.4 Unsteady Aerodynamics Model

One of the basic assumptions of the lifting line theory is that it is valid only for steady flows. That is, considering a fixed reference frame, the wing/blade is fixed and the wind field doesn't vary in speed and direction. This assumption can be easily violated for a wind turbine: the blades are rotating, and are usually subject to torsional deformations. Moreover, they experience added mass effects accelerating the fluid around them, flow separation, that is typical for real fluid characterized by viscosity, and rotational effects. In *QBlade* an *unsteady areodynamics model* is implemented and coupled with the *lifting line free vortex wake module*. This model consists of two parts, an attached flow model, and a Beddoes-Leishman dynamic stall model.

Attached flow model

The attached flow model takes into account the irrotational added mass effects. The fully attached contribution to the lift force is given by the lift generated on the aerofoil if it operates in fully-attached flow conditions at every angle of attack. The lift force is decomposed in three components:

- *irrotational lift*: is the lift force that arises in a irrotational flow as a reaction from the fluid accelerated by the aerofoil motion. The coefficient of lift for this component is expressed as

$$C_L^{NC} = \pi \frac{b_{hc}}{U_0} \dot{\alpha}_{str} \tag{2.87}$$

  being b_{hc} the half-chord length, U_0 the mean wind speed and $\dot{\alpha}_{str}$ the torsion rate of the aerofoil.

- *quasi-steady rotational lift*: corresponds to the lift force that would act on the aerofoil if the current motions are held constant for an infinite time. This is the steady lift generated by the aerofoil at the current angle of attack computed from the relative aerofoil motion, but without the influence of shed vorticity.

$$C_L^{QS} = C_L^{att}(\alpha_{qs}) \tag{2.88}$$

- *wake memory effect*: accounts for the influence of the shed vorticity in the wake on the quasi-steady angle of attack. In contrast to the classical formulation in BEM codes (the wake memory effect is modelled with an effective angle of attack, computed via step responses) the effective angle of attack is *directly* obtained from the free vortex wake formulation:

$$C_L^{circ} = C_L^{att}(\alpha_{eff}) \tag{2.89}$$

To obtain the quasi-steady angle of attack, first the influence of the shed vorticity on the angle of attack must be calculated considering the induction of the total shed vorticity in the vicinity of the blade, up to 8 chord lengths away from the trailing edge. This limitation is necessary to exclude the influence of the shed vorticity. Once obtained α_{shed}, it is used to compute the quasi-steady angle of attack:

$$\alpha_{qs} = \alpha_{eff} - \alpha_{shed} \tag{2.90}$$

This extra manipulation is necessary, because the common unsteady aerodynamics models are formulated for BEM codes, that uses indicial functions which are replaced by the free vortex wake model.

Dynamic Stall Model

Dynamic stall is a non-linear unsteady aerodynamics effect that occurs when aerofoils rapidly change the angle of attack: this causes the lift coefficient to follow an hysteresis loop, as reported in figure. This causes:

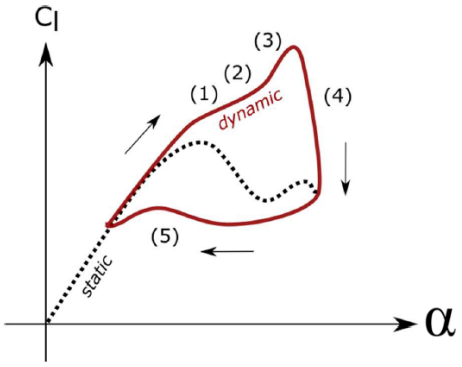


Figure 2.17: *Dynamic Stall Hysteresis cycle*

1. delay in the separation of the boundary layer;
2. shedding of a vortex from the leading edge of the aerofoil;
3. the vortex, that travel backwards above the wing containing high-velocity airflows, briefly increases the lift produced by the wing;
4. as it is convected behind the trailing edge, the lift reduces dramatically, and the wing is in normal stall;
5. the boundary layer shows a delay in the reattachment.

In *QBlade*, a Beddoes-Leishman dynamic stall model is implemented: the dynamic stall effect is modelled by means of three contributions:

1. *lagging potential flow lift*: the total lift computed through the potential flow theory (including rotational and irrotational contributions) is lagged through a first order low-pass filter with time constant τ_P, also referred to as *pressure time constant*:

$$\frac{dC_L^{lag}}{dt} = -\frac{U_0}{b_{hc}\tau_P}C_L^{lag} + \frac{U_0}{b_{hc}\tau_P}C_L^{pot} \tag{2.91}$$

2. *intermediate separation function*: models the dynamic circulatory lift and includes the separation point time lag:

$$C_L^{circ,dyn} = C_L^{att}(\alpha_{eff})f^{dyn} + C_L^{sep}(\alpha_{eff})(1 - f^{dyn}) \tag{2.92}$$

where f^{dyn} can be determined by solving

$$\begin{cases} \dfrac{df_{dyn}}{dt} = -\dfrac{U_0}{b_{hc}\tau_f}f^{dyn} + \dfrac{U_0}{b_{hc}\tau_f}f^{st}(\alpha*) \\ \alpha^* = \dfrac{C_{lag}}{\dfrac{\partial C_L}{\partial \alpha}} + \alpha_0 \end{cases} \tag{2.93}$$

permitting to assign a weight between the separated lift coefficient and the attached lift coefficient. α_0 is the angle of attack that returns a null linear steady lift and τ_f is the *boundary layer time constant*, that defines A low-pass filter for the separation function.

3. then the *vortex lift* contribution models the lift overshoot from the leading edge vortex, formed at the leading edge and convected over the aerofoil

$$\begin{cases} \dfrac{U_0}{b_{hc}\tau_v}\dfrac{dC_{L,vort}}{dt} + C_L = \dfrac{dC_v}{dt} \\ C_v = C_L^{circ,dyn}\left(1 - \dfrac{(1+\sqrt{f^{dyn}})^2}{4}\right) \end{cases} \tag{2.94}$$

Where τ_v is another low-pass filter time constant, the *vortex time decay constant*. Then the total lift, including attached and separated contribution is equal to

$$C_{L,dyn} = C_L^{circ,dyn} + C_L^{nc} + C_{L,vort} \tag{2.95}$$

The dynamic drag is evaluated from three contributions:

1. the steady drag at the effective angle of attack

$$C_D^{eff} = C_D(\alpha_{eff}) \tag{2.96}$$

2. the drag induced by the shed wake vorticity, using the *quasi-steady* angle of attack

$$C_{d,ind} = C_L^{circ,dyn} \cdot (\alpha_{qs} - \alpha_{eff}) \tag{2.97}$$

and the drag change induced by the separation delay:

$$C_{d,ind}^{f} = (C_D^{eff} - C_D(\alpha_0))\left[\frac{(1-\sqrt{f^{dyn}})^2}{4} - \frac{(1+\sqrt{f^{st}})^2}{4}\right] \tag{2.98}$$

Then the total drag is computed as:

$$C_D = C_D^{eff} + C_{D,ind} + C_{D,ind}^{f} \tag{2.99}$$

2.4.5 Tower Shadow

The model of tower shadow implemented in *QBlade* is an adaptation of the tower shadow model employed by AeroDyn[2]: is based on a superposition of the analytical solution for potential flow around a cylinder and a downwind wake model using a tower drag coefficient. The tower shadow model affects only velocity components in the x and y directions of the LSA frame, while the other components, perpendicular to tower cross section, remain unaltered.

2.4.6 Terrestrial boundary layer

The effect of the terrestrial boundary layer is modelled through the Hellmann power law:

$$U(z) = U_{hub}\left(\frac{z}{z_{hub}}\right)^{\alpha} \tag{2.100}$$

[2] `https://nwtc.nrel.gov/AeroDyn`

Chapter 3

Integration of the PoliTO floater model with *QBlade*

3.1 C++ library creation

3.1.1 Simulink model modification

To create an interface between the floater model and *QBlade*, the Simulink©(reported in Appendix B) model must be translated into C++ language, creating a *floater module* for *QBlade*. Before moving on to the code-generation phase, some modifications to the original Simulink© model are done:

- The array containing the positions, velocities and accelerations exiting from the *hull block* is sent to an output;
- Forces and moments exerted by the wind turbine come from an input;
- The initial conditions for the velocity to position integrator come from an input and must be set at each time step from the external environment;
- The instantaneous wave direction comes from an input;
- Froude-Krylov/diffraction forces, mooring forces, drag forces, radiation forces and hydrostatic forces are sent to five different outputs to be read by the *QBlade* post-processor.

After the code generation (see Section 3.1.2), these parameters will be accessible and modifiable, however, Simulink© doesn't allow some parameters to be external inputs. This problem regards:

- Tables containing the wave forces with varying wave direction and time and their relative size;
- Wave direction lookup table array and its relative size;
- Wave record time lookup table array and its relative size;
- Simulation time step;
- Stiffness, weight force and length of the mooring lines.

So, the generated code in principle is valid only for a particular wave, and the moorings' parameters are not modifiable. This is obviously a big limit that can be solved by modifying the produced code, with the objective to make it as flexible as possible. To do this, the open-source version of *Qt Creator*[1], a cross-platform integrated development environment was chosen.

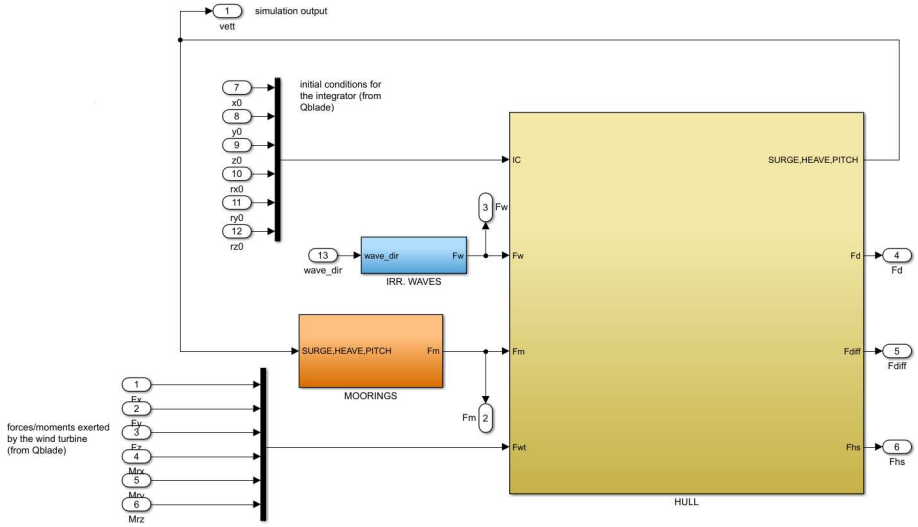


Figure 3.1: *Modified Simulink model for code generation*

3.1.2 Code generation and modification

Matlab© Simulink© Coder permits to choose various options for the code generation, the ones chosen are summarized in the following table:

System target file	`ert.tlc`
Language	C++
Toolchain	MinGW64 | gmake (64bit Windows)
Build Configuration	Faster Runs

Regarding the interface between the external environment:

Code replacement library	None
Shared Code placement	Auto
Support	floating point numbers non-finite numbers complex numbers absolute time continous time
Code Interface Packaging	C++ class

[1]Downloadable at `https://www.qt.io/download`

The code generated by the Simulink© Coder has a modular structure: `polito_floater.cpp` is the core of the *floater module*, while large variables are stored inside a separate source code, `polito_floater_data.cpp`. The headers contains the definition of functions, variables, classes and structures involved.

Model Files	`polito_floater.cpp`
	`polito_floater.h`
	`polito_floater_private.h`
	`polito_floater_types.h`

Data Files	`polito_floater_data.cpp`
Utility Files	All Matlab© libraries source codes and headers needed to run the model

Table 3.1: *Files generated during the code generation phase*

Some of these files need a further elaboration. In particular, all the tables containing the wave force records are stored inside the `polito_floater_data.cpp` file, as well as the arrays used in the prelookup tables. Also the time step of the simulation is fixed, when it should instead come from *QBlade*. These issues make the generated code not flexible, not having the possibility to simulate the behaviour of the system with different wave forces and different mooring lines parameters. To eliminate these problems, the code has been modified.

The main part of the code is the source file `polito_floater.cpp`, that contains all the mathematical operations performed by the model translated into C++ code, in form of different member functions (*"the methods"*) contained inside a C++ class called `polito_floater_ModelClass`: in particular two of these functions, `void initialize()` and `void step()` need to be modified in the definition, as well as in the declaration. The first is the initialization entry point of the code, and is useful to set all the simulation parameters that will be fixed during run-time, while the second is the output entry point, that contains the instructions to effectively run the model, producing all the results at each time-step. There's also another function, `void terminate()` that contains the destructor, a member function that is called whenever the run has to be interrupted or terminated. Its main purpose is to free the resources (memory allocations, open files, etc...) which were acquired by the object "floater" during its life and/or deregister from other entities which may keep references to it.

The above functions have storage class `public`, meaning that they can be accessed from anywhere within the class or outside the class. Other functions contained inside the code and declared as `private`: they can be accessed only by the functions inside the class and are not useful to interface with the code. Also the input and output parameters are declared as `public` and grouped together inside `polito_floater_ModelClass`.

In a few words, the declared class is the bridge between the object "floater" and the external world, that can see *only* the public members of the class, once all the headers are included, basically ignoring how they are internally composed. The details about the modification of the produced code are furnished in Appendix C.

3.1.3 DLL Creation

A Dynamic-Link Library (DLL) is a software library that is loaded dynamically by a software at run-time, without being statically connected to the executable during the compilation phase. The main advantages of using such libraries is that they are loaded (linked) by the software only if they are needed, and that, in case of new versions or bug-fixes, the software that uses the DLL can be upgraded by simply replacing the file, paying attention if the functions inside the DLL are changed in their declaration or definition. A Dynamic-Link Library is loaded by the operative system (OS) in the memory space of the process that requires it, ensuring performances that are practically the same of the software which uses it, and can be thought as a collection of functions that are *exported* to the DLL during the building process.

The link against the executable is done, in this work, *implicitly*, meaning that it's managed directly by the linker during the compilation phase of the executable, assuming that the DLL is always present inside the system. Each time, from the source code of *QBlade*, a function contained inside the DLL is called, the linker links the call to the function to a fictitious *stub* function. Inside the executable, there is a table containing the *stubs* to all the DLL functions: during the executable loading, the OS automatically loads the needed DLL mapping the *stubs* to the entry point of the relative DLL function. To make a function available inside a DLL, it must be *exported* to it, specifying the storage class adding the keyword `__declspec(dllexport)` after the type of the function and before its name.

The open source version of Qt Creator gives only the possibility to develop 32-bit applications or libraries, due to the presence of 32-bit debugger and compiler. So, before starting creating the DLL, a new 64-bit compiling environment (compiler-debugger) must be installed, besides a 64-bit version of the *Make* build automation tool used by Qt, *qmake.exe*[2].

After the installation, a new Kit must be configured in Qt Options panel manually adding the new compiler, the new debugger and the new Qt Version (*qmake*). After these operations, a new *Qt Creator* project is started, with the template *"Library/C++ Library"*. The DLL developed in this work is called *Floaty*. At the beginning only a few files are present: one source file, `floaty.cpp`, and two header files, `floaty.h` and `floaty_global.h`. In the `Floaty.pro` files all the source codes and the headers generated by Matlab© are added specifying the directory and the name of the file, together

[2]The complete package used for this work is *qt-5.5.0-x64-mingw510r0-seh-rev0*, downloadable from `https://sourceforge.net/projects/qt64ng/`. It includes 64-bit versions of qmake.exe, and MinGW, a complete compiling environment.

with the needed Matlab© libraries.

All this elements are included in the *floaty* main directory in form of sub-directories only for convenience.

Project Template	C++ library
Chosen Kit	MinGW x64
C Compiler executable	gcc.exe
C++ compiler executable	g++.exe
Debugger executable	gdb.exe
Required Qt Modules	QtCore
Class Name	floaty
Header File	floaty.h
Source File	floaty.cpp

Table 3.2: *Main Settings for the DLL creation with Qt*

If `polito_floater_ModelClass` represents a bridge to the external world for the *floater module*, `floaty.cpp`, the source file of the library, is the road on which informations travel. It must give to the user access to all variables/structures produced as outputs during the execution of the code, functions needed to run the code during the execution, together with the possibility to set all available parameters before starting simulating. Practically, it acts like a control panel for the *floater module*. This is done by defining three functions to be exported to the DLL:

- `initialize_floater`, useful to initialize the parameters of the simulation such as the time step, the moorings' stiffness, length and weight force, the number of timesteps and the number of directions considered to discretize the round angle (that receives as arguments), and to call the initialize function contained inside the C++ class `polito_floater_ModelClass` passing all the matrices containing the wave forces, the directions considered and the time array of the wave records;

- `platform`, that collects the results produced after that the simulation goes on by a time step and calls the step function of the C++ class `polito_floater_ModelClass`. In particular, when called, function `platform` collects all the results inside a structure of type `fromModel` that can be used to access the platform kinematics and dynamics as sub-fields. It must be called passing as arguments the wave direction, forces and moments exerted by the wind turbine on COG of the complete structure and the initial conditions for the speed-to-position integrator.

- `close_floater`, that calls the destructor from the C++ class `polito_floater_ModelClass` when the simulation finishes or is stopped. It doesn't receive any argument.

Naturally, these functions and variables must be declared in the header file `floaty.h`. For coding details, the full source code of *floaty* is provided in Appendix 3.1.2.

`initialize_floater`	Set Simulation Timestep Set Number of Directions Set Number of wave timesteps Set Mooring lines Weight Force Set Mooring lines Stiffness Set Mooring lines length Set Mooring lines pre-tensioning Set integrator initial conditions
`platform`	Platform kinematics Mooring Forces Wave Forces Drag Forces Radiation Forces Hydrostatic Forces Send WT forces/moments to floater
`close_floater`	Calls the Destructor

Table 3.3: *DLL functions*

Once the interface is created, the DLL is ready to be build in Release mode. The product of the build is a *.dll* file, *Floaty.dll*, ready for the integration in *QBlade*.

3.2 Interfacing with *QBlade*

3.2.1 Defining a coherent reference frame

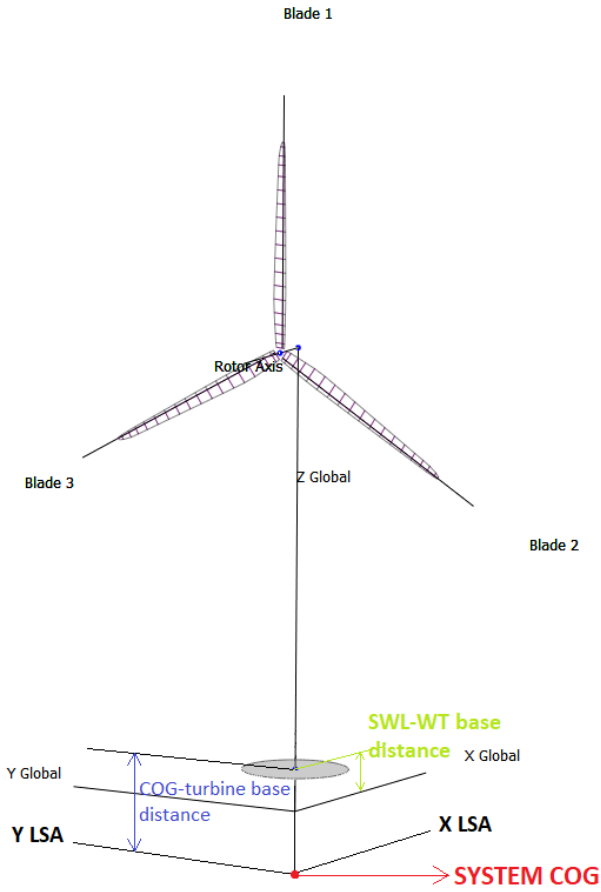


Figure 3.2: *Reference frames defined in QBlade*

To interface correctly with *QBlade*, there's the need to introduce both the FRA and the LSA frames on it. The reference frame defined in *Qblade* is right-handed and by the default, is set to have its origin at the base of the wind turbine, the "X Global" axis pointing downwind and the "Z Global" axis pointing along the wind turbine's tower. The Global reference frame of *QBlade* is inertial, being its axes fixed, so, to interface with *Floaty*, the wind turbine base is shifted upwards by a quantity equal to the distance, measured in the FRA frame, between the still water level line and the turbine base, making the Global reference frame coincident with the FRA reference frame and so the X Global - Y Global plane corresponding to the still water plane. When the *floater module* is active, the wind turbine base automatically shifts upwards if the SWL-WT base distance is greater than 0. *QBlade* evaluates the Thrust vector in the Global reference frame: then moments exerted by the wind turbine on the COG of the floater are calculated in the LSA frame taking into account the distance between the floater's COG and the wind turbine base, and the rotations/translations of the LSA frame.

3.2.2 Setting up the interface

All the headers related to *floaty* are moved into a unique folder `src/floaty` , into the project directory of *QBlade*, together with the .dll file of the library. After it, *Floaty.dll* is included in `qblade_v09.pro` file, together with the related headers. To access the function declared in the header `floaty.h`, that are the ones exported to the DLL, from a particular source file (*.cpp*) contained inside the *QBlade* source code it's sufficient to use the preprocessor instruction `#include "floaty/floaty.h"`. This is done at the beginning of the source file `QLLTSimulation.cpp`. Once done it, all the DLL's function are normally accessible by simply typing their name. In the *QBlade* header file `QLLTSimulation.h`, the structure `fromModel floater` is declared to import all the variables from the *floater module*.

A new member function, `OnLoadFloaterParameters()` is created: it opens a directory (pre-specified by the user, see Section 3.2.3) and loads the files:

- `surge.dat`
  `sway.dat`
  `heave.dat`
  `roll.dat`
  `pitch.dat`
  `yaw.dat`
  containing the wave force matrices,
- `time.dat` containing the time array of the wave force record,
- `deg.dat` containing the directions array.

After initializes the *floater module* calling the function `initialize_floater`:

```
1 initialize_floater(getTimeStep(), num_deg, num_wave, m_moorW,
    m_moorK, m_moorL, preT, time, deg, surge_force, sway_force,
    heave_force, roll_force, pitch_force, yaw_force);
```

where

- `geTimeStep()` is the time step used by *QBlade*
- `num_deg` is the number of directions used to discretize the round angle;
- `num_wave` is the number of elements contained in the wave force records;
- `m_moorW` is the single mooring line weight force;
- `m_moorK` is the mooring lines stiffness;
- `m_moorL` is the mooring lines length;
- `preT` is the mooring lines pre-tensioning;

- `time` is the time array of the wave force record;
- `deg` is the directions array;
- `surge_force`
  `sway_force`
  `heave_force`
  `roll_force`
  `pitch_force`
  `yaw_force`
  are, respectively, the matrices containing the wave forces and moments in the different directions.

In the member function `onStartAnalysis()` an if condition is added: if the *floater module* is active, first the function `OnLoadWaveForces()` is called and data are acquired. Then *QBlade* proceeds in its operations, calling another member function, `calcHAWTresults()`. Again, if the *floater module* is active, the moments exerted by the wind turbine on the COG of the structure are calculated:

$$M_{r_x} = -T_y^{LSA}\ t_z^{AP}$$

$$M_{r_y} = T_z^{LSA}|t_x^{AP}| + T_x^{LSA}|t_z^{AP}|$$

$$M_{r_z} = -T_y^{LSA}|t_x^{AP}| + T_x^{LSA}t_y^{AP}$$

Where

- M_{r_x} = `mx.last();`
  M_{r_y} = `my.last();`
  M_{r_z} = `mz.last()`
  are the moments exerted by the wind turbine on the COG of the floater;
- T_x^{LSA} = `ThrustLSAx.last()`,
  T_y^{LSA} = `ThrustLSAy.last()`,
  T_z^{LSA} = `ThrustLSAz.last()`
  are the thrust components respectively in the x, y and z directions of the LSA frame;
- t_x^{AP} = `m_ThrustActingPoint.data()->x;`
  t_y^{AP} = `m_ThrustActingPoint.data()->y;`
  t_z^{AP} = `m_ThrustActingPoint.data()->z`
  are the coordinates of the thrust acting point respectively on the x and z directions, in the LSA frame.

Note that the thrust components in the LSA are obtained inside the member function `CalcHAWTResults()` taking the thrust vector `thrustfloaty` in the FRA, and rotating it using the rotation matrices implemented in *QBlade*. The x, y, z components are then stored inside `ThrustLSAx`, `ThrustLSAy`, `ThrustLSAz`.

```
...

    //thrust in the LSA frame
    if(active_floater)
    {
        thrustfloaty.RotX(m_PlatformRollAngleX*PI/180);
        thrustfloaty.RotY(m_PlatformPitchAngleY*PI/180);
        thrustfloaty.RotZ(m_PlatformYawAngleZ*PI/180);
        ThrustLSAx.append(thrustfloaty.x);
        ThrustLSAy.append(thrustfloaty.y);
        ThrustLSAz.append(thrustfloaty.z);
    }

...
```

then, the `platform` function is called assigning the returning values to the structure `floater`

```
...

floater =  platform(wave_direction, m_ThrustX.last(), m_ThrustY.
    last(), m_ThrustZ.last(), mx.last(), my.last(), mz.last(),
m_PlatformTranslation.x, m_PlatformTranslation.y, swl_cog_floaty,
    m_PlatformRollAngleX*PI/180, m_PlatformPitchAngleY*PI/180,
    m_PlatformYawAngleZ*PI/180);

...
```

Where

- `m_PlatformTranslation.x`,
  `m_PlatformTranslation.y`,
  `swl_cog_floaty`
  are the instantaneous coordinates (x, y, z) of the floater COG in the FRA,

- `m_PlatformRollAngleX*PI/180`,
  `m_PlatformRollAngleX*PI/180`,
  `m_PlatformRollAngleX*PI/180`
  are the rotations r_x, r_y, r_z, in radians, around the axes of the FRA.

once the function is called, `floater` contains all the dynamic and kinematic variables. *Qblade* provides, besides a graphical 3D rendering of the wind turbine, also the possibility to plot the variables present in the class that is in use. For this reason, for each variable that exits from the floater module, a new array in the class `QLLTSimulation` is created to be visible to all the member functions, and after the function `platform` is called, the values contained in `floater`, at each time step, are memorized in the last position of each of these arrays. The member function `PrepareOutputVectors()`

will give the possibility to create plots and export them as .csv files (commma-separated values) for the following dynamic and kinematic variables coming from the *floater module*:

- instantaneous COG coordinates in the FRA and floater rotations, (angular) speed and (angular) acceleration (angular) in all the directions,
- hydrodynamic drag forces and moments in all the directions,
- mooring forces and moments in all the directions,
- hydrostatic forces and moments in all the directions,
- diffraction and Froude-Krylov Forces in all the directions,
- radiation forces in all the directions,
- thrust force in LSA frame,
- moments exerted by the wind on the turbine in the LSA. Together with the variables already present in the standard version of *QBlade*.

3.2.3 Graphical User Interface

To make the user able to change the *floater module* input parameters, a graphical user interface is created inside the *QLLT Simulation* module of *Qblade*, in form of a control panel called *"Floater Settings"* where the user can:

- activate or deactivate the *floater module* through a radio button;
- choose the directory containing the wave force matrices, the wave record time array and the directions array in form of `.dat` files;
- set the initial conditions for the *floater* module, that is:
  - COG x, y, z coordinates in the FRA in m
  - Floater rotations around the FRA axes in deg
- insert the COG-turbine base distance in m
- edit the number of time steps composing the wave force records
- edit the number of directions considered
- set the incoming wave direction in deg
- set the moorings Stiffness in N/m
- set the moorings Weight Force in N
- set the moorings length in m
- set the moorings pre-tensioning in N

The creation of that interface (see Figure 3.3) makes the user able to simulate a very large range of conditions for the FOWT, giving the possibility to perform free-decay analysis by simply using, in this first version of the DLL, 0-filled matrices for the wave force records. To enable the *floater settings* control panel to communicate with the *QLLT Simulation module*, the declaration of the member function `QLLTSimulation` is modified to receive as arguments all the parameters set by the user.

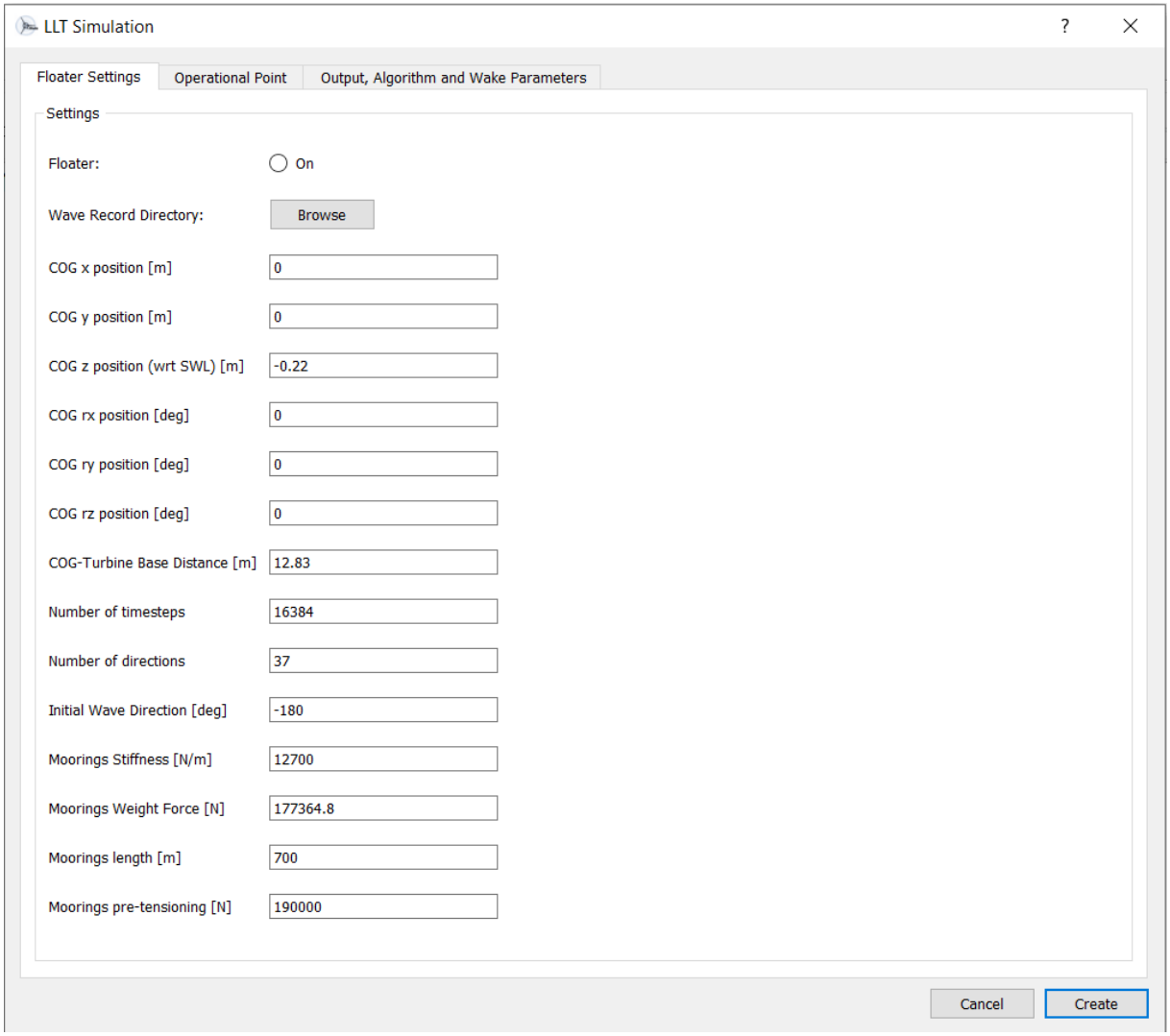


Figure 3.3: *"Floater Settings" control panel*

Chapter 4

System

The system object of the coupled simulations *QBlade-floater module* is composed by four main elements, as shown in Figure 4.1:

- NREL 5 MW wind turbine
- Fincantieri Seaflower floater
- Mooring lines
- Surrounding Environment

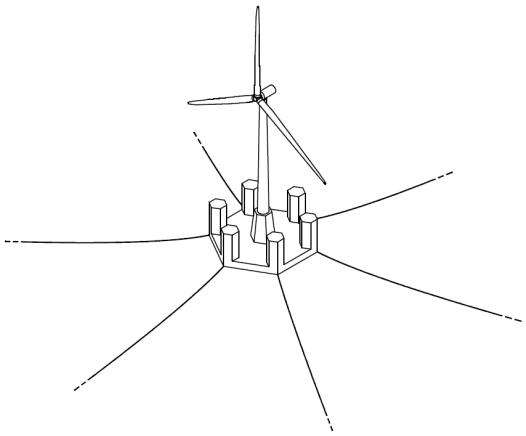


Figure 4.1: *Isometric representation of the system*

4.1 Seaflower floating platform

The floater, designed by Fincantieri[22] "*consists of a hexagonal submerged platform acting as main buoyant body and damper and six semi-submerged columns at the corners that fullfill the static and dynamic lateral stability requirements while ensuring high transparency to wave motions*". The main scope of this project is to overcome the high costs of bottom-fixed foundations

Platform diameter	63m
Platform height	4.5m
Columns diameter	10.3m
Central column base diameter	14.5m
Columns height	18m
Floater mass	15,400t
Floater density	600 kg/m^3

Table 4.1: *Characteristics of the Seaflower Floater.*

construction and installation when dealing with high sea depths (50-200 m) and to endure the challenging waves and winds of highly energetic sites such as the North Sea. Fincantieri's choice of marine concrete as material, without any bracing, makes the Sea Flower earn cost effectiveness and durability, strikingly critical in the aggressive marine environment. The essential data about the floater are reported in the following table:

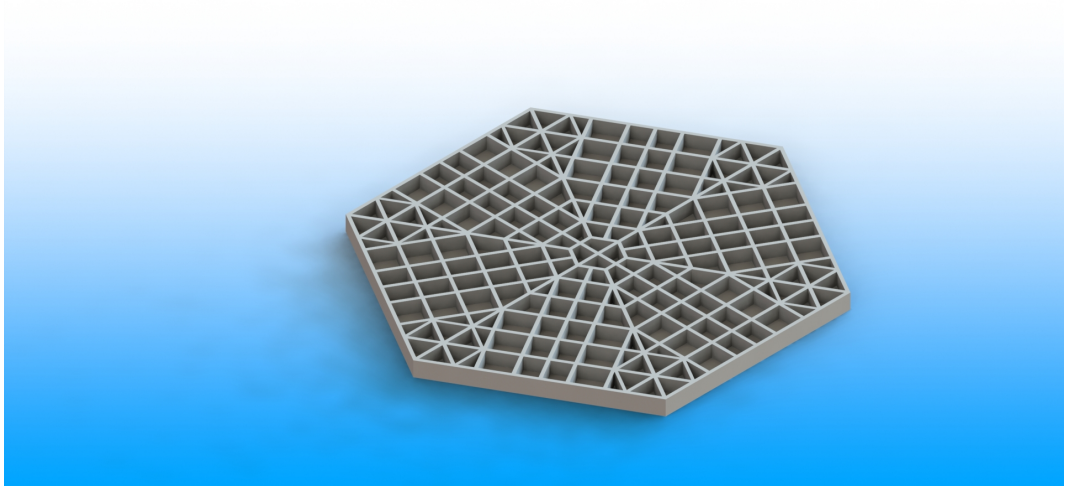


Figure 4.2: *Internal structure of the Seaflower floating platform*

4.2 NREL 5 MW

The U.S. Department of Energy's National Renewable Energy Laboratory's (NREL) Wind department has developed a standardized wind turbine for usage in different investigations, named *offshore 5-MW baseline wind turbine* [23]. Since the floater is designed for a wind turbine of 5 MW nominal power and NREL5MW is completely and publicly defined, this was the best choice.

Rated Power	5MW
Rotor Orientation	upwind
Rotor Diameter	$126m$
Hub Diameter	$3m$
Cut in wind speed	$3m/s$
Rated wind speed	$11.4m/s$
Cut out wind speed	$25m/s$
Rotor Overhang	$5m$
Shaft Tilt	$5°$
Precone	$2.5°$
Rotor mass	$110t$
Nacelle mass	$240t$
Hub inertia about rotor axis	115.926 kgm^2
Tower height	$87.6m$
Distance nacelle base/rotor axis	77.6m
Hub Height	$90m$

Table 4.2: *NREL offshore 5MW baseline wind turbine specifications*

4.2.1 Wind Turbine Specifications

The wind turbine is designed with a power rating of 5MW, resulting from considerations on offshore floater feasibility and state of the art for wind turbines at the time it was initiated. The design is the result of specifications from several different precious prototypes, such as *MultibridM5000* and *REPower 5M*, and studies, such as *WindPACT*, *RECOFF* and *DOWEC*. The rotor radius is 63m, while the hub height was chosen to be 90 m, which is quite low with respect to the rotor diameter(126 m): however, this minimizes the overturning moment generated by the thrust acting in the rotor ensuring a 15 m clearance between the blade tips at their lowest point and an estimated extreme 50-year wave height of 30m. The remaining specifications are summarized below in Table 4.2.

4.2.2 Blades Modelling

In *QBlade*, using the *Aerofoil desing* and the *HAWT Rotorblade Design* modules, there's the possibility to design, starting from zero, the rotor of a wind turbine. Using the publicly-defined specifications provided by Jonkman [23], each aerofoil is imported in *QBlade* using the *foil file* format (.dat) specifying the foil coordinates, and then the structure of the *NREL5MW baseline* wind

Y(m)	Chord	β[°]	Aerofoil
1,5	3,2	13,08	DU99W350LM
2,86	3,54	13,08	DU97W300LM
5,6	3,85	13,08	DU91W2250LM
8,33	4,17	13,08	DU91W2250LM
11,75	4,55	13,08	DU93W210LM
15,85	4,65	11,48	DU93W210LM
19,95	4,46	10,16	NACA64618
24,05	4,25	9,01	NACA64618
28,15	4,01	7,80	NACA64618
32,25	3,75	6,54	NACA64618
36,35	3,50	5,36	NACA64618
40,45	3,26	4,19	NACA64618
44,55	3,01	3,13	NACA64618
48,65	2,76	2,32	NACA64618
52,75	2,52	1,53	NACA64618
56,17	5,31	0,86	NACA64618
58,90	2,09	0,37	NACA64618
61,63	1,40	0,16	NACA64618
63,00	0,7	0	NACA64618

Table 4.3: *Geometrical properties and aerofoils used to model the rotorblade in QBlade*

turbine rotor is modelled using 19 radial substations, the characteristics of each one given in Table 4.3, where Y refers to the coordinate of the nth radial substation measured from the hub center and β to the *twist angle*, the angle by which the aerofoil is rotated around the Y axis of that particular substation. The 3D view of the blade model can be seen in Figure 4.3

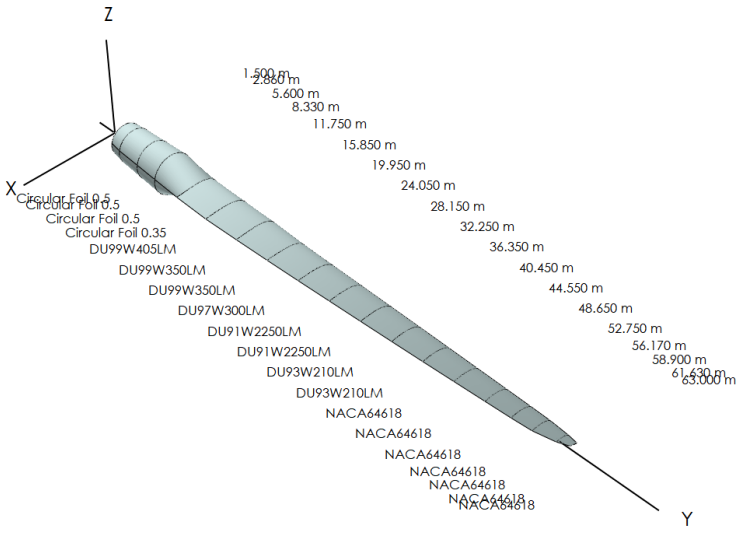


Figure 4.3: *NREL5MW rotorblade, with type of aerofoil used and coordinates for each radial substation*

4.3 Moorings

The mooring system is designed in order to prevent the system from drifting under the action of waves, current and wind, and to increase rotational stability. The solution implemented by Fincantieri is a spread inert catenary mooring system consisting of a set of six pre-tensioned lines evolving radially from each vertex of the hexagonal floater, uniformly distributed on the 360° of the water plane, to keep the floater in position. Two different sea depths are taken into account by Fincantieri (50 and 200 m).

The single lines are composed by three segments. Starting from the anchor at the seabed, these sections are:

1. Metal Chain
2. Polyester rope (section of higher length)
3. Metal chain

This solution gives a good stiffness while guaranteeing light lines: its properties are summarized in Table 4.4. To use the elastic model described in Section 2.3.1, the stiffness of each line has to be calculated. Observing that the segments composing the mooring lines can be considered as a series of springs, the equivalent stiffness is:

$$\frac{1}{k_{eq}} = \frac{1}{k_{c1}} + \frac{1}{k_{cr}} + \frac{1}{k_{c2}}$$

where, in this work, $k_{c1} = k_{c2} = 6690\ kN/m$ is the stiffness of the two, identical chains and $k_r = 12.7\ kN/m$ is the stiffness of the polyester rope. The line

Number of mooring lines	6	
Angle between adjacent lines	60°	
Seabed depth	50 m, 200 m	
Mooring leg composition	rope-chain-rope	
Unstretched lengths (50 m depth)	100-500-100 m	
Unstretched lengths (200 m depth)	100-700-30 m	
Pre-tensioning	190 kN	
Chain	type	studlink, R3
	diameter	90 mm
	mass per unit length	182 kg/m
	minimum breaking load	6647 kN
	axial stiffness	699 MN
Rope	type	polyester
	diameter	160 mm
	mass per unit length	16.8 kg/m
	minimum breaking load	7112 kN
	axial stiffness	59.3 MN

Table 4.4: *Mooring system properties*

equivalent stiffness is $k_{eq} = 12.65\ kN/m$, very close to the stiffness of the rope alone.

The line weight is considered constant and, for each line, equal to that of a single 100m chain. In the computation, the mooring lines are considered to be rectilinear in every instant: the rope segment of the line is predominant in length and is tensioned by the two chain segments, stiffer and heavier. The angle between lines and seabed is so low that the catenary curves are close to the linearity, and, due to the large length of the mooring lines, their configuration is considered to be fixed, considering the typical motions of the floater.

Thanks to the linearity assumption, the moorings can be considered as linear springs with the bottom end fixed at the anchor points and the top end connected to the floater.

4.4 Complete System

The fixed reference axes frame *Oxyz* (FRA), dashed line, in Figure 4.4, is defined as a right-handed reference frame with the origin O identified as the intersection between the system vertical axis of "simmetry" at rest and the mean water plane, the z axis pointing upwards and the x axis pointing backwards, downwind. This frame is fixed in space and thus represents an inertial reference. The local structure axes frame *Gxyz* (LSA), continous line in Figure 4.4, has its origin in the system center of gravity G and its axes, at the starting position, are parallel to those of the FRA. The LSA is used to describe the motions of the system in the FRA.

Mass	16000 t
Displacement	15608 m^3
Draft	12 m
Center of gravity from SWL	2.33 m
Roll radius of gyration	21.91 m
Pitch radius of gyration	21.90
Roll radius of gyration	20.38

Table 4.5: *Characteristics of the complete system.*

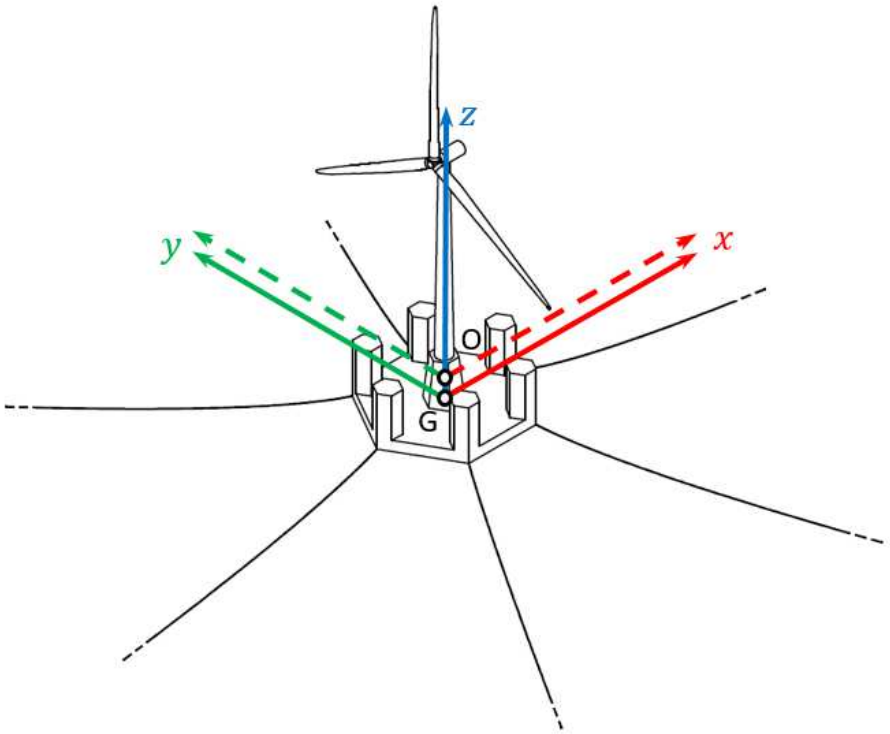


Figure 4.4: *Fixed Reference frame Oxyz (dashed line) and local structure axes Gxyz (continous line)*

Chapter 5

Results of the coupling

5.1 Onshore-Offshore Comparison

In this section, the differences between an onshore and an offshore installation are examined. Two simulations are performed:

- *onshore case*: the *floater module* is deactivated, and the simulation is carried on with a value for the wind speed at the hub center $U_{hub} = 17m/s$.

- *offshore case*: the *floater module* is active and the simulation is carried on for a value of incoming wave direction $\theta = 0°$, with $U_{hub} = 17m/s$, $H_s = 3.28m$ and $T_e = 10.54s$. The effect of the terrestrial boundary layer is taken into account by means of the Hellmann exponent $\alpha = 0.16$, and the incoming wind direction γ is the same of the incoming wave direction.

In the figures, the red curves refer to the *onshore case*, while the blue curves to the *offshore case*.

5.1.1 Effects on the wake

As the floater moves, it influences actively the wake produced by the wind turbine, changing its shape and modifying the flow field in a determinant manner. In Figure 5.1 is presented the flow field on a plane coincident with the xz plane of the FRA, in the *onshore case*, while in Figure 5.2 in two planes, one parallel to the xy plane and shifted by a quantity $z_{hub} = 90m$ on the z axis, and on the xz plane. The wake has a uniform aspect, being the zone of largest magnitude of the velocity vector the ones influenced by the wing tips and the blade roots, where vortex strength rapidly increases to model the trailing vorticity. The flow field, being decelerated by the power extraction operated by the wind turbine, cause the wake to decelerate far from the rotor. It shrinks due to the modelled terrestrial boundary layer: the parts of the wake at an higher altitude tend to be convected faster with respect to the parts closer to the soil. In the *offshore case*, see Figure 5.3, the wake presents a much larger turbulence with respect to the *onshore case* and for this reason

tends to maintain the same shape far from the rotor, and the trailing vortices tend to be deformed. It's worth noting how the wake *tilts* upwards, due to the effect of the pitch angle.

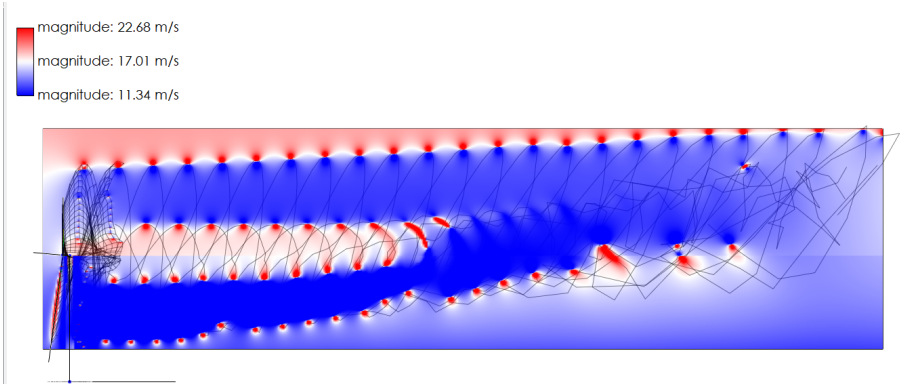


Figure 5.1: *Wake in the onshore case, xz cut plane*

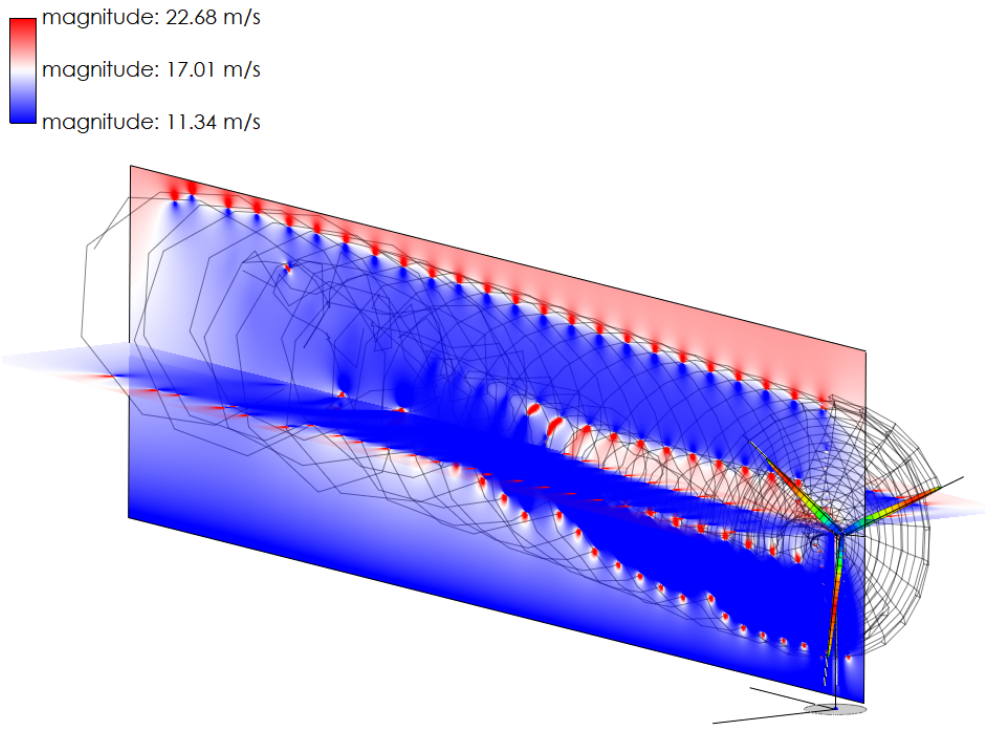


Figure 5.2: *Wake in the onshore case, perspective view*

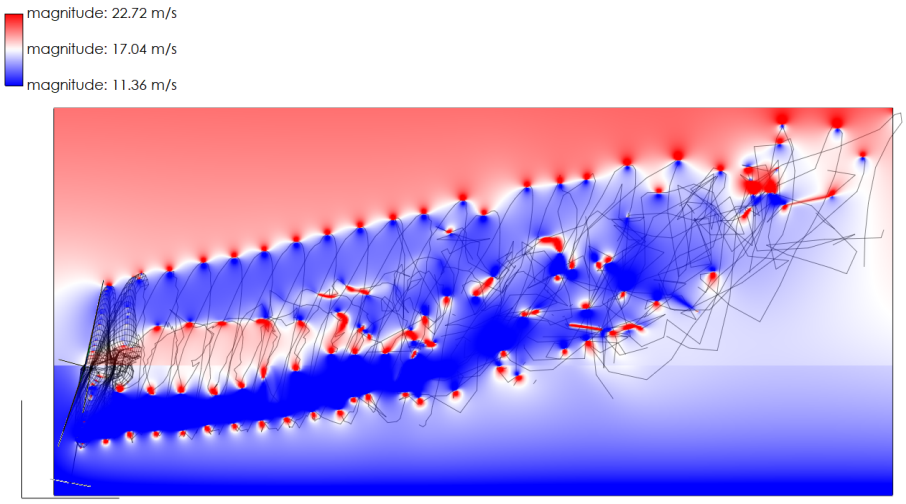


Figure 5.3: *Wake in the offshore case, xz cut plane*

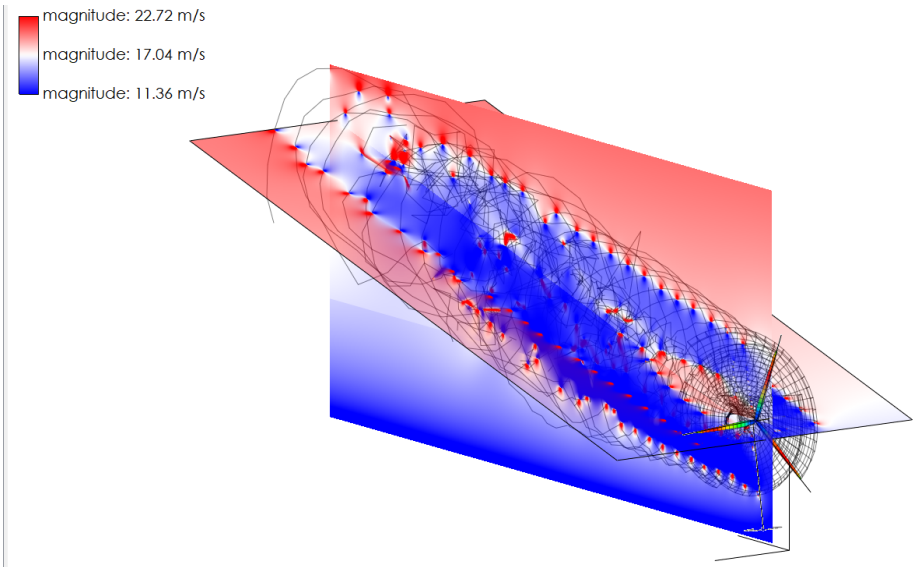


Figure 5.4: *Wake in the offshore case, perspective view*

5.1.2 Effects on the aerodynamics

Power Coefficient

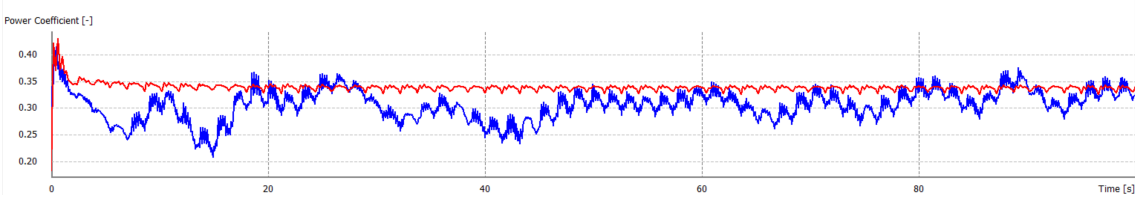


Figure 5.5: *Power coefficient as a function of time, in the onshore/offshore cases, first 100s*

The fact that the FOWT is moving under the action of waves and wind

influences different parameters. The mean power coefficient decreases: the instantaneous power coefficient for the offshore case is below the one for the onshore installation for great part of the simulation time (see Figure 5.5), showing also a much more fluctuating behaviour, due to the fact that the floater's motions cause the wind turbine to work at an efficiency lower than the nominal one.

Lift to Drag ratio

The lift to drag ratio C_L/C_D (Figure 5.6), in the *offshore case*, starts to have a different behaviour with respect to the *onshore case*, showing different peaks reaching values close to double the corresponding one for the *onshore* case. The effect should be further investigated: it is surely caused by the newly-introduced floater dynamics, and is always associated with a decrease of the angle of attack.

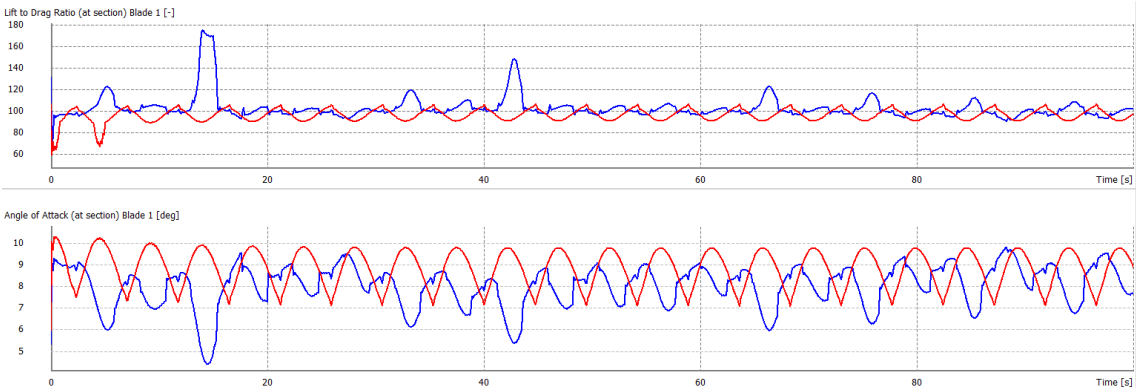


Figure 5.6: *Lift to drag ratio and angle of attack for blade 1 as a function of time, in the onshore/offshore cases, first 100s*

Velocity Induced from Tower

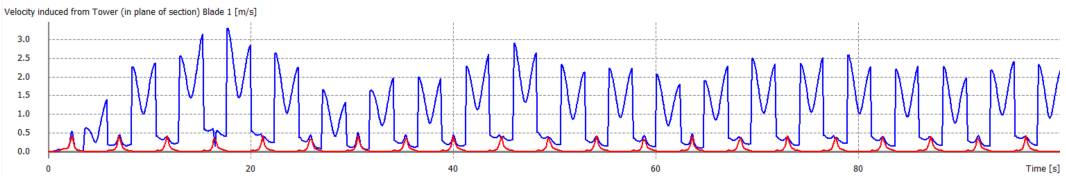


Figure 5.7: *Velocity induced from tower as a function of time, in the onshore/offshore cases, first 100s*

Velocity induced from tower (Figure 5.9) in the *offshore case* shows an important rise, that can be related to the acceleration, induced by the floater's motion, of the fluid around it.

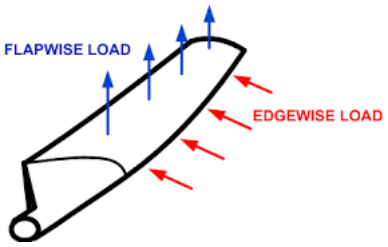


Figure 5.8: *Loads causing the root bending moments.*

Root Bending Moments

The root bending moments, M_f and M_e (flap-wise and edge-wise components) caused by the loads imposed by the wind in the edgewise and flapwise directions, shows larger oscillations in the *offshore* case,especially for the flapwise component. This can led to stronger oscillations of the wind turbine blades, causing larger fatigue stresses.

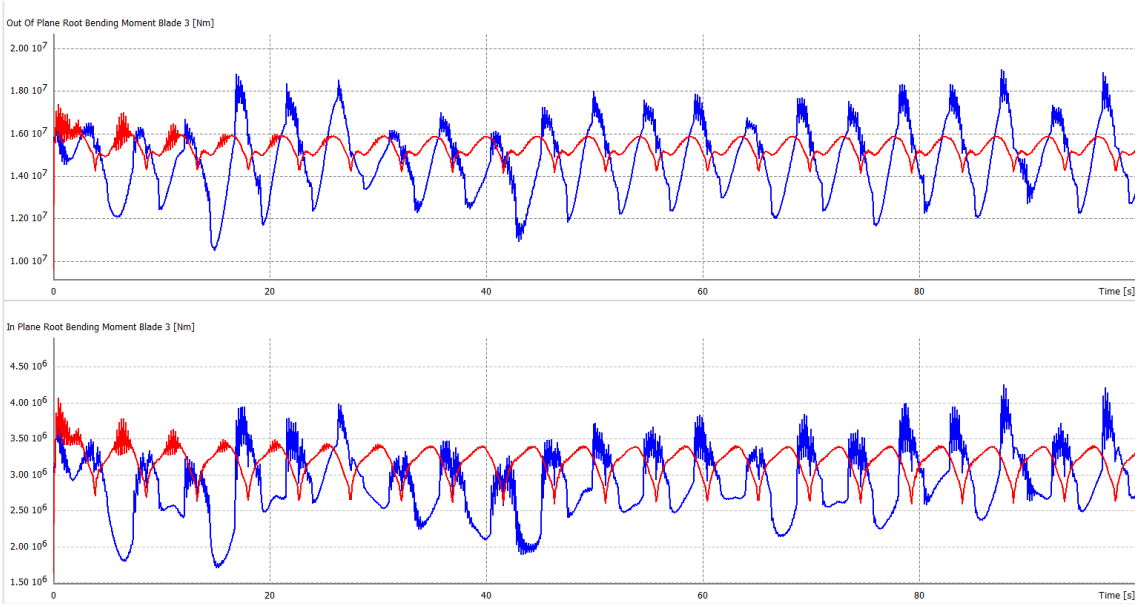


Figure 5.9: *Out of Plane (Flapwise) and In-Plane (Edgewise) root bending moments for blade 3 as a function of time, in the onshore/offshore cases, first 100s*

5.1.3 Normal and Tangential Loads on the Blades

Normal and tangential loads on the blades starts to have a much more irregular behaviour, and the tangential force decreases, as can be seen in Figure 5.10.

Thrust

The floater dynamics affects the thrust in all the directions (Figure 5.11). In particular, the x component decreases, followed by an increase in the y

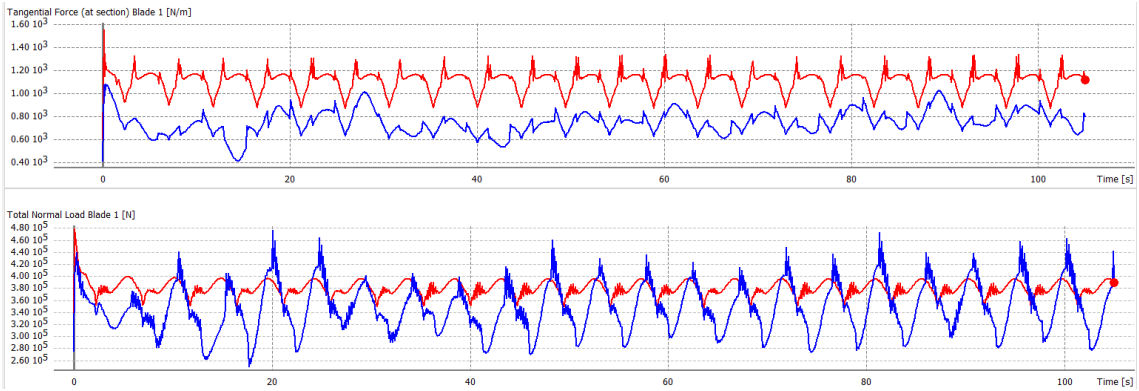


Figure 5.10: *Normal and tangential loads for blade 1, first 100s*

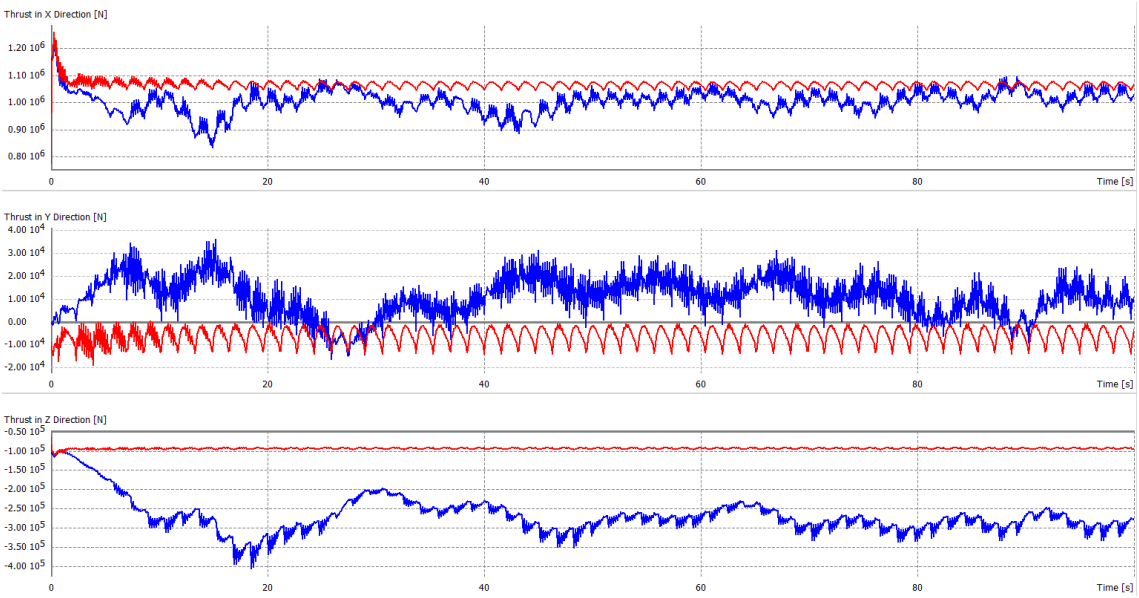


Figure 5.11: *Thrust in the x, y and z directions of the FRA frame as a function of time, in the onshore/offshore cases, first 100s*

component and a decrease in the z component. This last one can be considered as a source of damping for the system, indeed it tends to push downward the center of gravity of the structure. However, the behaviour of the total Thrust (Figure 5.12) is much more oscillated in the offshore case.

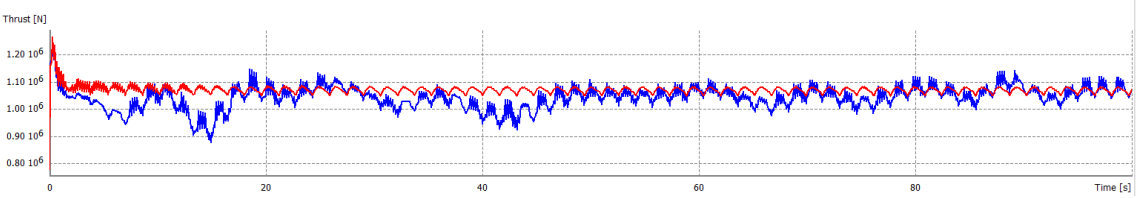


Figure 5.12: *Total thrust as a function of time, in the onshore/offshore cases, first 100s*

5.2 Test Cases

In this section the behaviour of the system is examined for different test cases, with different values of wind speed, wave height and period, examining its behaviour on its operative range, when the wind turbine is actively producing power, and when the turbine is off, that is, when the environmental conditions are such that the system must be shut down to avoid damages. This test cases should be considered as a preliminar investigation of the behaviour of the system and a verification of the correct coupling between *Floaty* and *QBlade*. In each test case, a unique value of H_s, T_p and wind speed at the hub, U_{hub} is used, while the wave and wind directions, respectively θ and γ varies covering the right angle from 0° to 90° with no relative angle. A uniform wind field is considered, with the presence of an atmospheric boundary layer, which is taken into account by means of the Hellman power law.

F_τ and F_n denote, respectively, the tangential and the normal loads on the blades. **Notice that** in all the graphs, instead of the instantaneous coordinate of the COG in z direction of the FRA, is reported the instantaneous z coordinate of the turbine base in the FRA. The two representations are equivalent, being the COG z coordinate obtainable by subtracting the distance between the COG and the turbine base, equal to 12.83m.

	θ	Colour
	°	
Case 1	0	Red
Case 2	22.5	Green
Case 3	45	Blue
Case 4	67.5	Black
Case 5	90	Magenta

Table 5.1: Colours Legend

5.2.1 Normal Operating Conditions

In this test case, the system is considered as subject to both aerodynamic and hydrodynamic loads, with values under the normal operating conditions of the FOWT. The Hellman's power law exponent is $\alpha = 0.16$ (neutral air above flat, open coast) with reference height $z_{hub} = 100.5m$, that is the hub height in the FRA reference frame.

The values of $H_s = 3.28m$, $T_e = 10.54s$ and $U_{hub} = 17m/s$ are chosen to be inside the normal operating conditions of the system, as specified by Fincantieri.

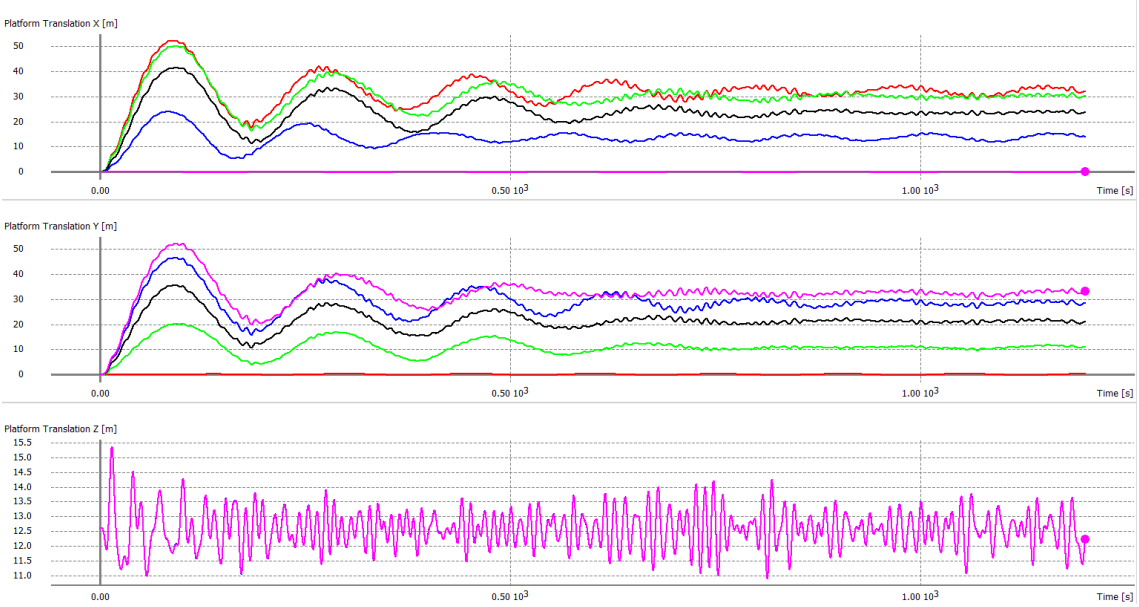


Figure 5.13: *Translations, normal operating conditions*

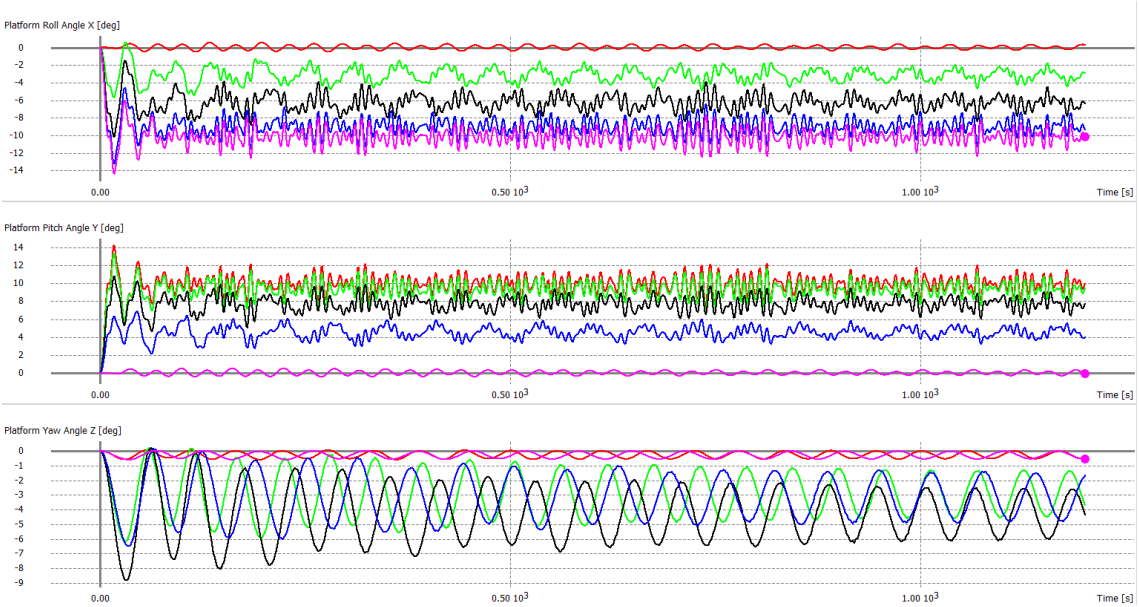


Figure 5.14: *Rotations, normal operating conditions*

Kinematics As expected, surge and sway show an opposite trend: when $\theta = 0°$, sway motions are negligible, whereas surge is clearly dominant. By increasing θ, sway becomes more and more relevant, until the situation is reversed when the incoming wave/wind direction is parallel to the y axis. An equilibrium condition is reached when $\theta = 45°$, when both surge and sway has a comparable trend. Differences arise mainly due to the different values of added mass and drag coefficients for the two directions. In particular, as can be seen in Figure 5.15, the instantaneous values of the hydrodynamic drag forces are different, also if comparable. A similar situation is present also for pitch and roll. Heave is insensible to the variation of θ: this is due to the

fact that both aerodynamic and hydrodynamic loadings in this direction do not depend on the incoming wave/wind direction, and heave itself does not depend on other floater motions.

While surge, sway, roll and pitch reaches their maximum values at the extremes of the interval [0°,90°], yaw reaches its peak value for $\theta = 45°$, mainly thanks to the contribution of the aerodynamic yaw moment (Figure 5.17) that has a negative mean value, maximum for this value of θ.

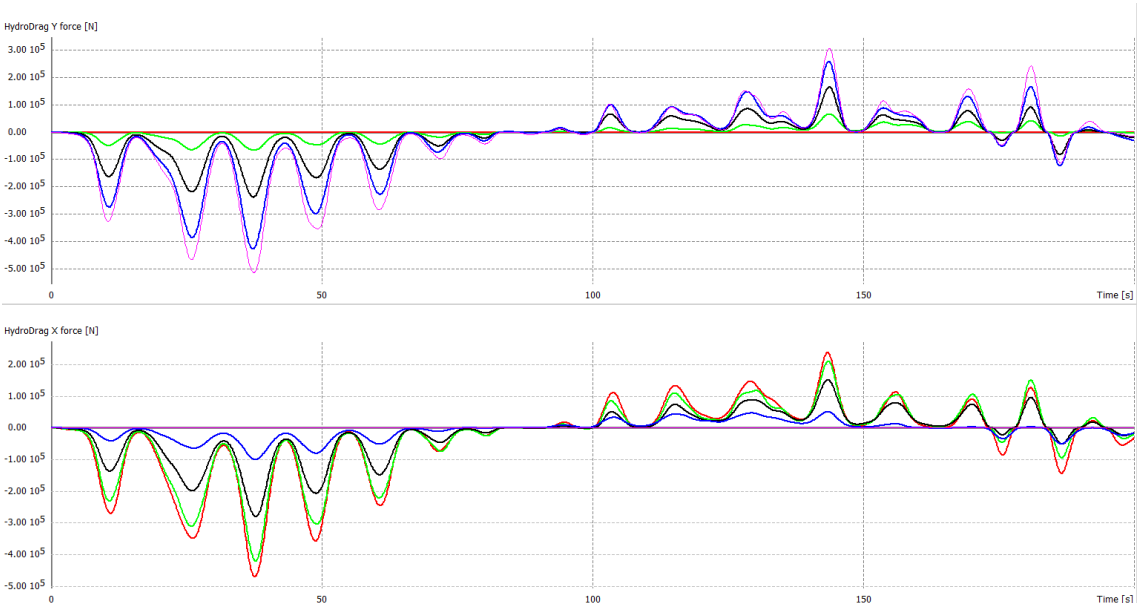


Figure 5.15: *Hydrodynamic drag force in surge and sway*

Aerodynamics and Blade Loads As already stated in Section 5.1, introducing the floater dynamics impacts on the aerodynamic parameters, on the power coefficient and on the blade loads. Aerodynamic variables starts to have a much more irregular behaviour, and the power coefficient reduces. What it's important to notice is that changing the wave/wind direction produces a very small impact: neglecting local oscillations curves are very similar, being practically superposed. A comparison between the maximum and mean values of the aerodynamic parameters in the Normal Operating Conditions and at the Upper Threshold of operating conditions can be found in Tables 5.3 and 5.4.

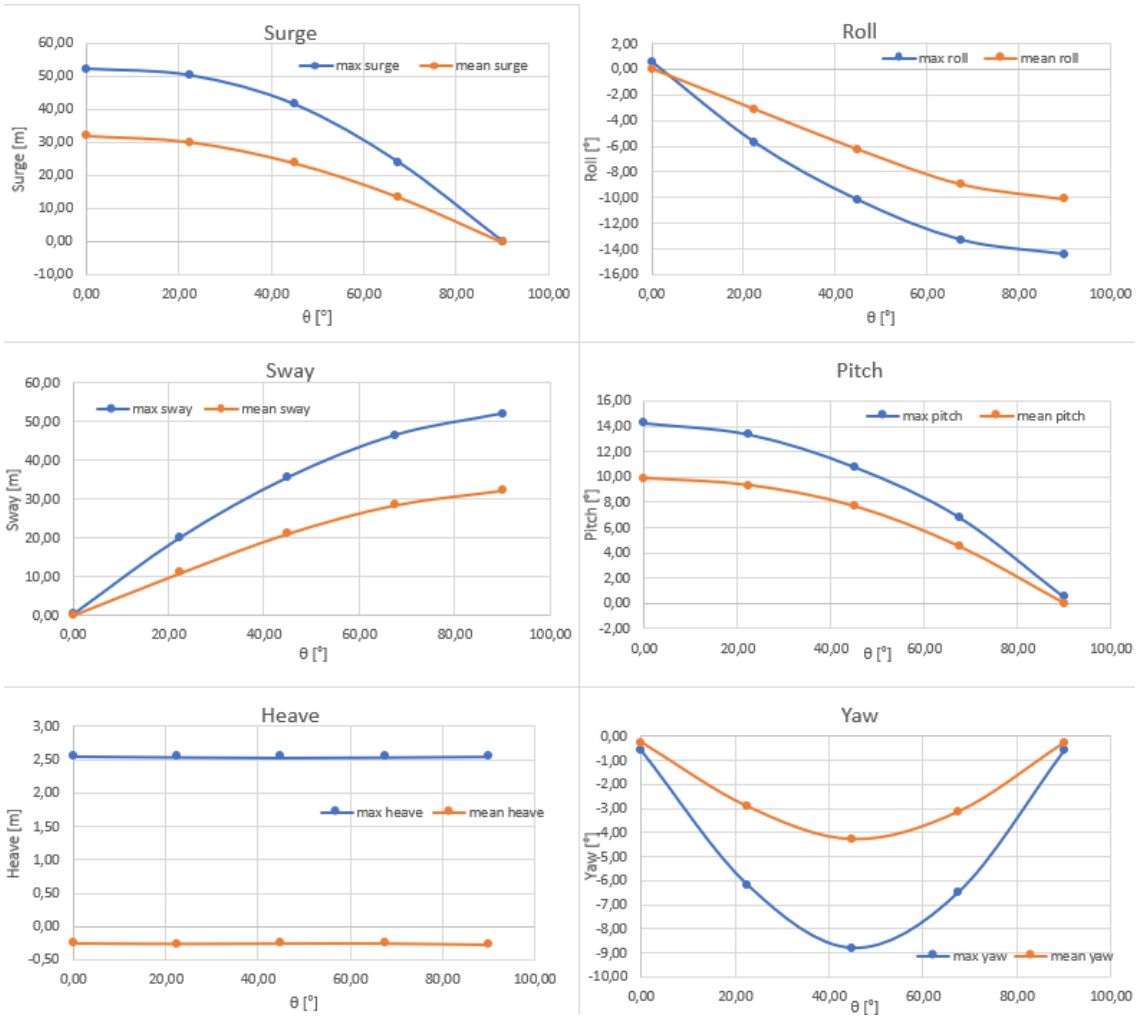


Figure 5.16: *Peak and mean values of floater's motions, normal operating conditions*

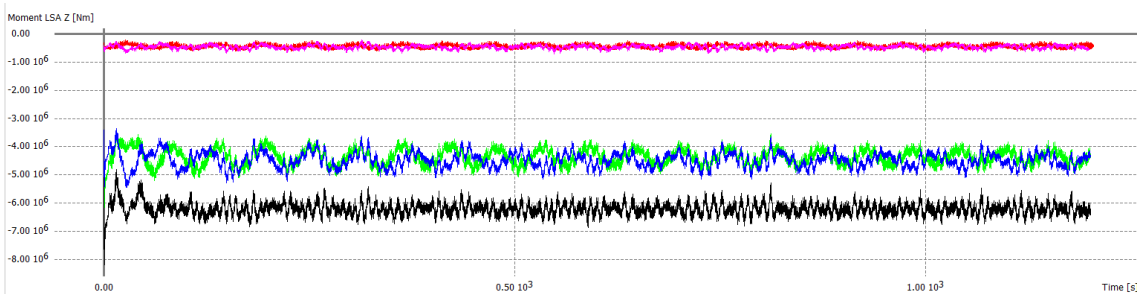


Figure 5.17: *Aerodynamic Yaw moment in the LSA frame, normal operating conditions*

5.2.2 Upper Threshold of operating conditions

In this test case, the system is considered as subject to both aerodynamic and hydrodynamic loads, with values set on the upper threshold of the operating conditions of the FOWT. The Hellman's power law exponent is $\alpha = 0.16$

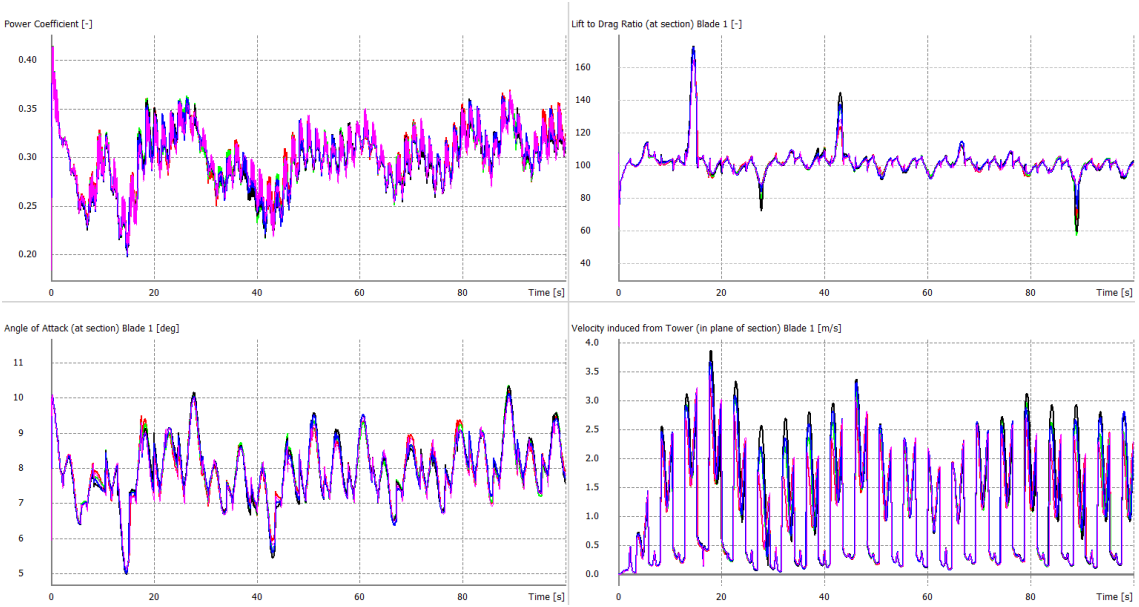


Figure 5.18: *Power coefficient, C_L/C_D ratio, angle of attack and velocity induced from tower, normal operating conditions, first 100s.*

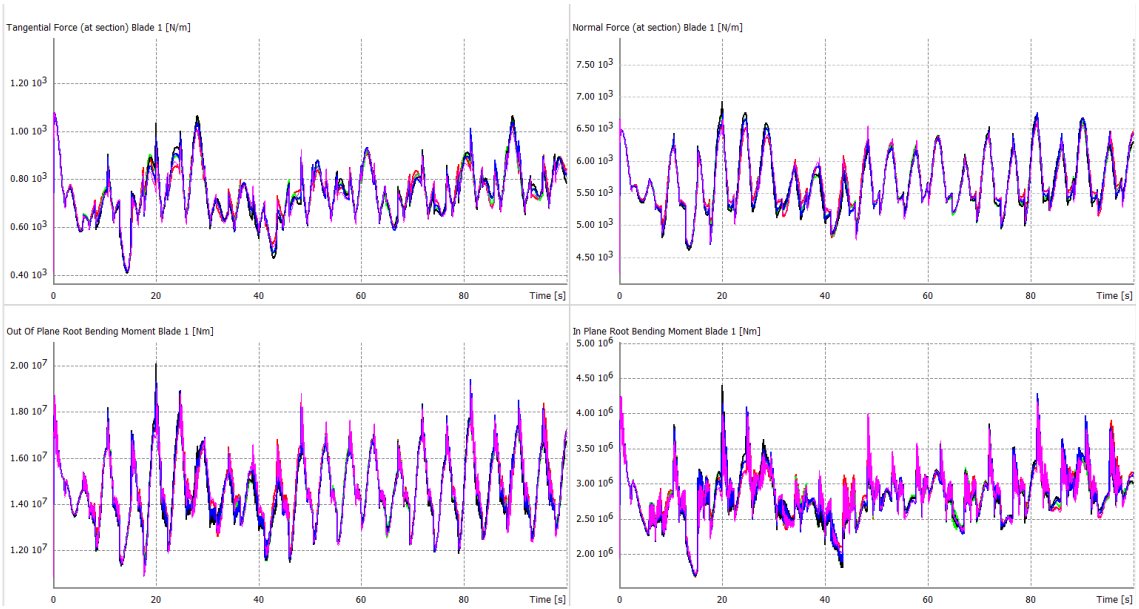


Figure 5.19: *Blade loads in normal operating conditions, first 100s.*

(neutral air above flat, open coast) with reference height $z_{hub} = 100.5m$.

The values of $H_s = 6.5m$, $T_e = 10s$ and $U_{hub} = 25m/s$ are chosen to be at the upper threshold of the operating conditions, as specified by Fincantieri.

Kinematics Also in this case surge and sway, as well as pitch and roll, assume their largest values respectively for $\theta = 0°$ and $\theta = 90°$, with the difference that the magnitude of the floater's motions is much larger than the previous case. Heave and yaw are decoupled from the other DOFs. The curves

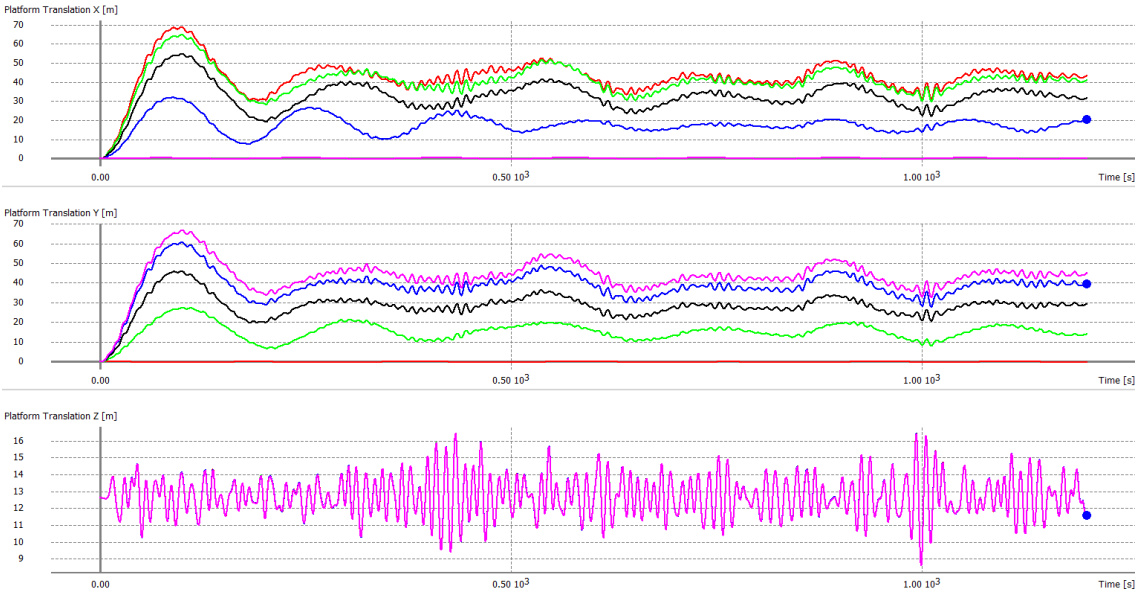


Figure 5.20: *Translations, upper threshold of operating conditions*

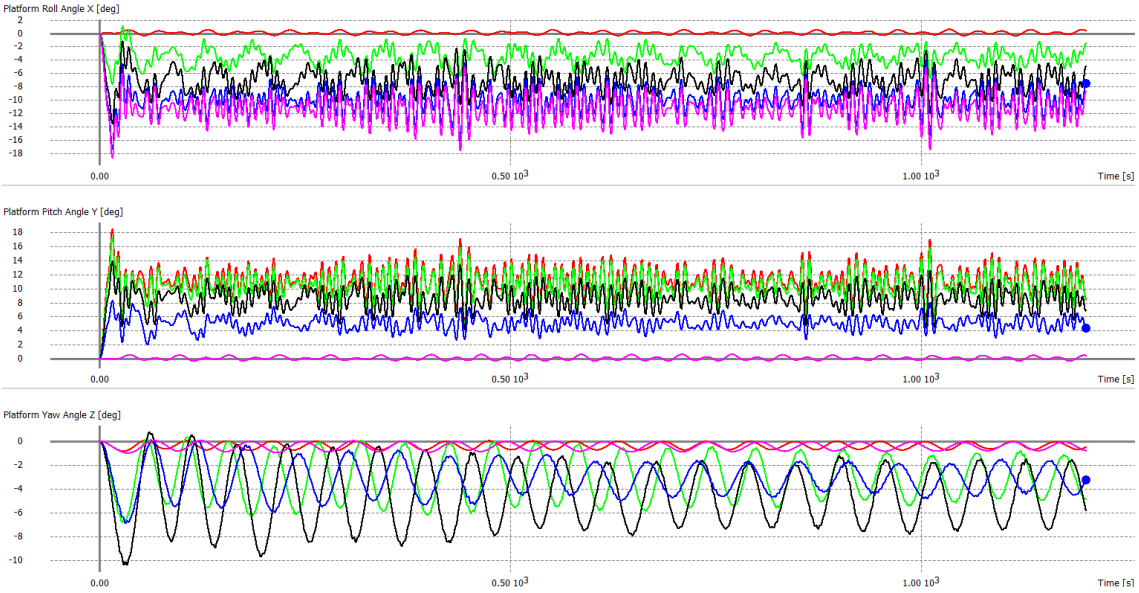


Figure 5.21: *Rotations, upper threshold of operating conditions*

show similar trends.

Aerodynamics and Blade Loads As already discussed in Section 5.2.1, the only effect on the aerodynamic parameters and on the blade loads caused by the floater is the drastic modification of the shape of the curves, that, varying the wave/wind incoming direction, remains practically equal. In this case, the power coefficient shows a much more important decrease, and blade loads amplifies. A comparison between the maximum and mean values of the aerodynamic parameters in the Normal Operating Conditions and at the

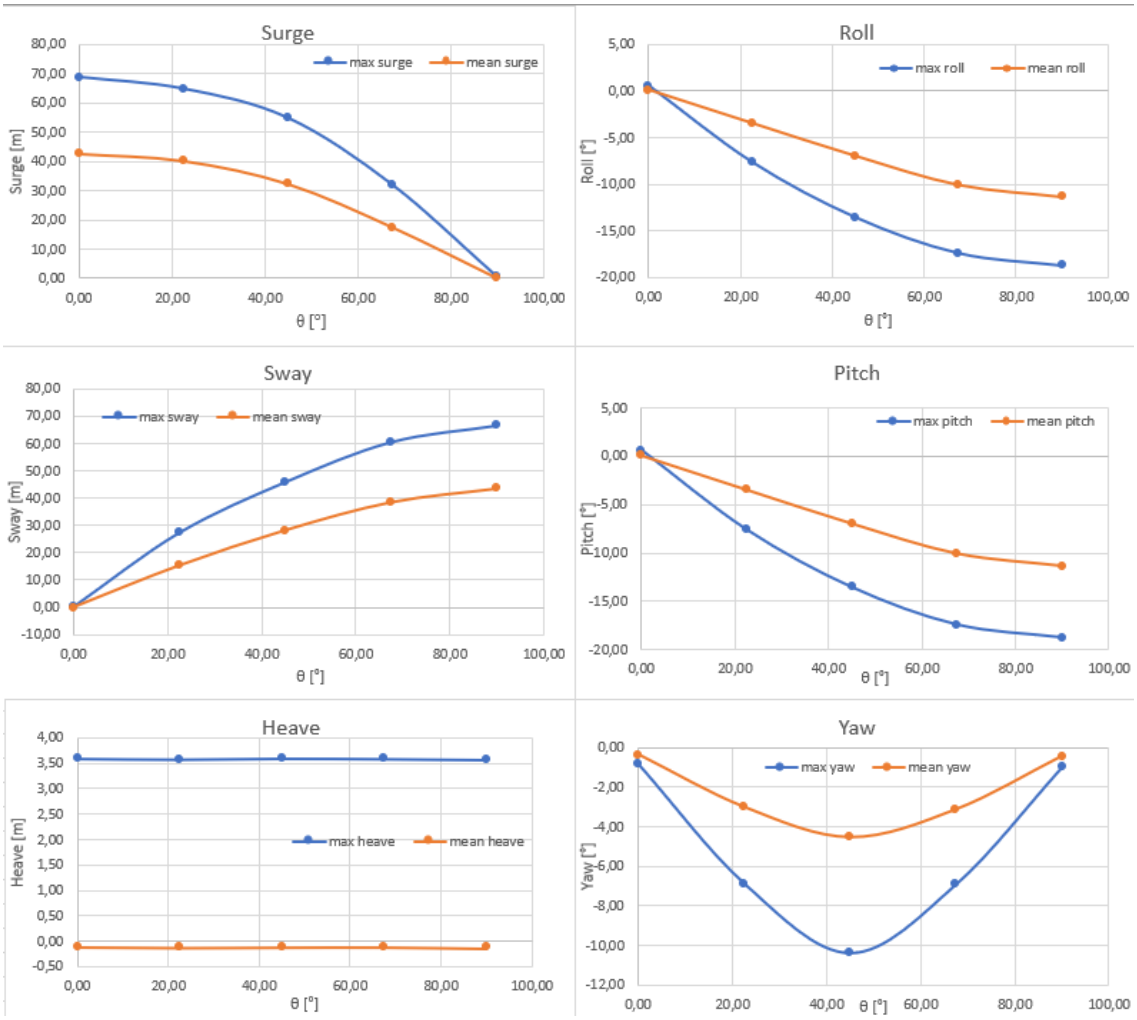


Figure 5.22: *Peak and mean values of floater's motions, upper threshold of operating conditions*

Upper Threshold of Operating Conditions can be found in Tables 5.3 and 5.4. It is noticeable that both the maximum and mean values of the angle of attack α are lower in Normal Operating Conditions than at the Upper Threshold: the probability of stall is thus larger in the last case, and the C_L/C_D ratio reduces dramatically.

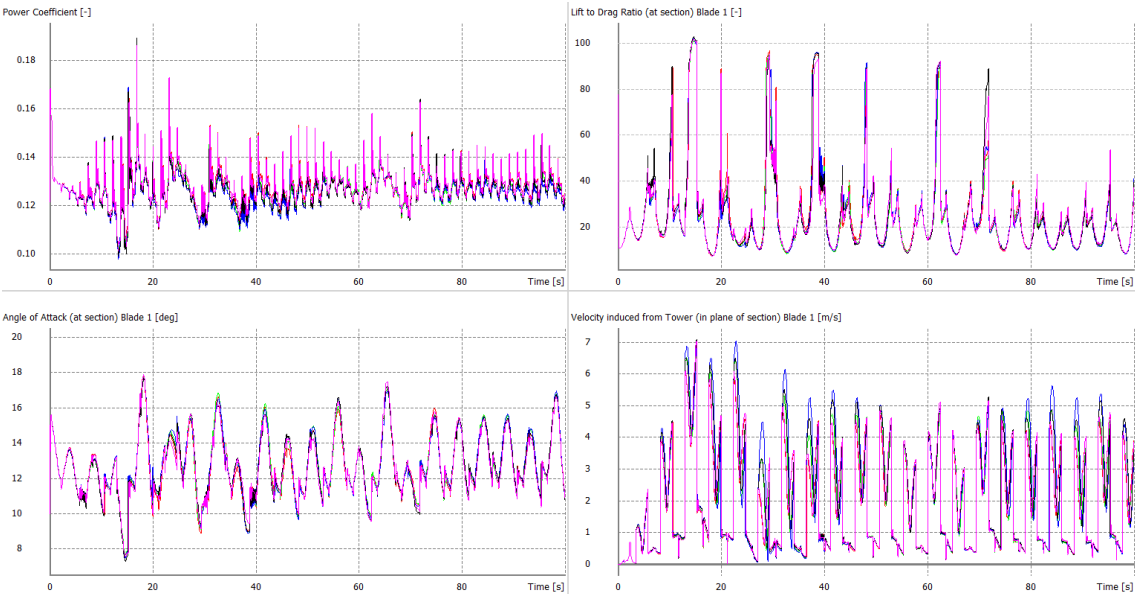


Figure 5.23: *Power coefficient, C_L/C_D ratio, angle of attack and velocity induced from tower, upper threshold of operating conditions, first 100s.*

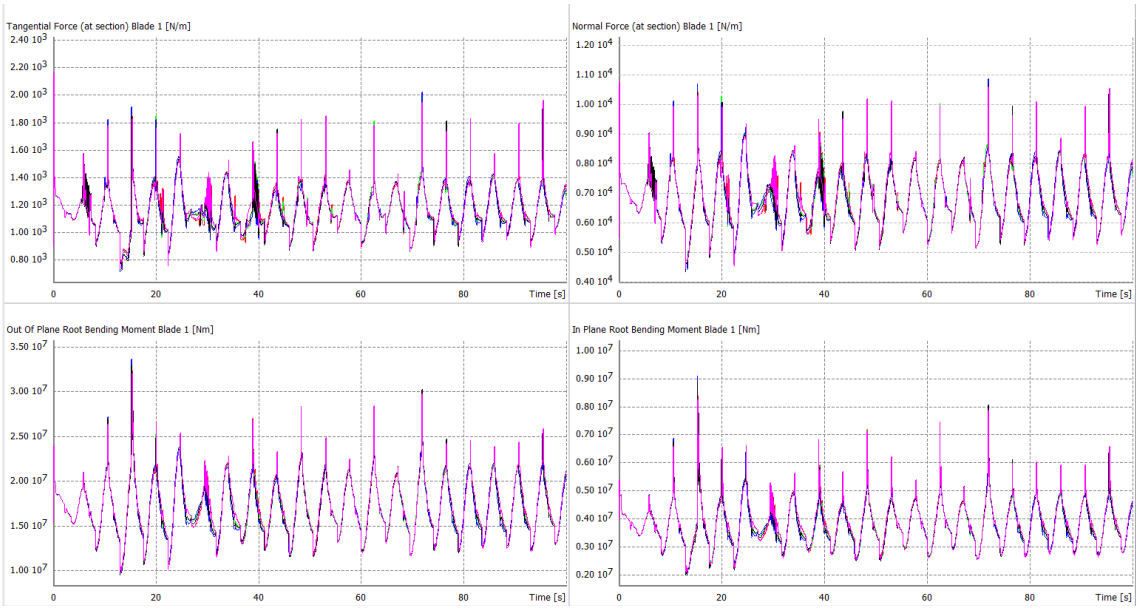


Figure 5.24: *Blade loads in upper threshold of operating conditions, first 100s.*

5.2.3 Extreme wave conditions

In this case, that is the one of a 50 years return period storm, the wind turbine is inactive due to the the fact that a wind speed equal to $U_{hub} = 40m/s$, well above the cut-off wind speed of the generator, is chosen. Waves with significant wave height $H_s = 13m$ and period $T_e = 13.4s$ lash the floater. This time the nacelle yaw is considered to be fixed and equal to 0°, while wind/wave direction varies in the range [0°,90°] with no relative angle, as in the previous cases.

Kinematics Even if the floater's motions are not so much different than the case of normal operating conditions, the fundamental difference is on the accelerations, that, in this case, are much larger: in Table 5.2 are reported the maximum values, in absolute value, of the floater's accelerations in the two cases.

Aerodynamics and blade loads Being the nacelle yaw fixed, loads on the blades show a larger variability with respect to the previous case. Both the edgewise and the flapwise components of the root bending moment are maximum when the incoming wind/wave direction is equal to $\theta = 67.5°$, while all loads are minimum when $\theta = 90°$, due to the fact that the nacelle yaw is fixed to the $\theta = 0°$ direction: so, the surface that the wind "sees" drastically reduces. The tangential force exerted on the blade shows the largest values for $\theta = 67.5°$. Mean and maximum values of loads on the blades as a function of the incoming wave/wind direction, for this case, can be found in Table 5.5.

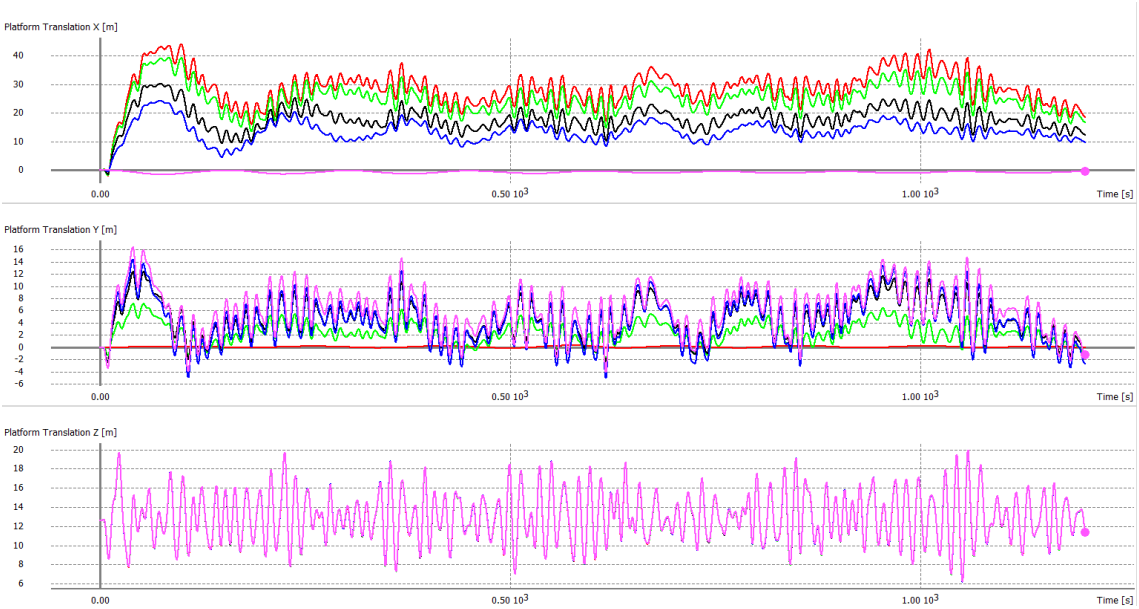


Figure 5.25: *Translations, extreme wave conditions*

	Normal Operating Conditions						***Extreme Wave Conditions***					
θ	x	y	z	r_x	r_y	r_z	x	y	z	r_x	r_y	r_z
$°$	m/s^2	m/s^2	m/s^2	$°/s^2$	$°/s^2$	$°/s^2$	m/s^2	m/s^2	m/s^2	$°/s^2$	$°/s^2$	$°/s^2$
0	0.52	0	0.54	0.02	1.15	0	1.83	0	1.65	0	4.06	0
22.5	0.48	0.20	0.54	0.46	1.08	0.14	1.70	0.79	1.65	1.69	3.79	0.30
45	0.37	0.35	0.54	0.82	0.84	0.21	1.27	1.49	1.65	2.88	3.01	0.38
67.5	0.19	0.46	0.54	1.07	0.47	0.15	0.72	1.92	1.65	3.75	1.64	0.29
90	0	0.51	0.54	1.17	0.03	0	0	2.05	1.65	4.08	0.02	0

Table 5.2: *Acceleration in Normal Operating Conditions and in Extreme Wave Condtions*

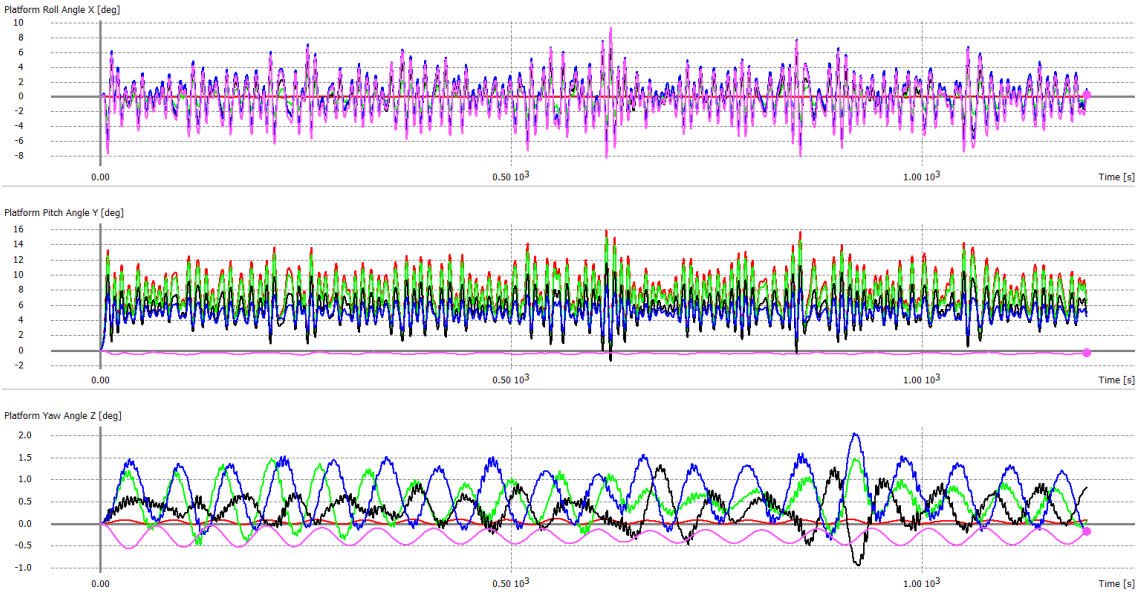


Figure 5.26: *Rotations, extreme wave conditions*

Mean Values	C_p	C_l/C_d	α	F_τ	F_N	M_f	M_e
	-	-	°	N	N	Nm	Nm
Normal Operating Conditions	0.32	98,85	8,27	7,62e4	3,53e5	1,50e7	3,00e6
Upper Threshold of O.C.	0.13	21.54	13,21	9,51e4	4,08e5	1,74e7	3,90e6

Table 5.3: *Mean values of aerodynamic parameters and loads on the blades in normal operating conditions and at the upper threshold of operating conditions*

Max Values	C_p	C_l/C_d	α	F_τ	F_N	M_f	M_f
	-	-	°	N	N	Nm	Nm
Normal Operating Conditions	0.41	172,76	11,36	1,31e5	5,13e5	2,04e7	4,74e6
Upper Threshold of O.C.	0.19	101,11	19,83	2,70e5	8,39e5	3,37e7	9,65e6

Table 5.4: *Maximum values of aerodynamic parameters and loads on the blades in normal operating conditions and at the upper threshold of operating conditions*

θ	F_τ	F_N	M_e	M_f
0	4,05e+04	2,56e+05	8,19e+06	1,06e+06
22.5	3,51e+04	2,61e+05	8,41e+06	1,03e+06
45	2,33e+04	2,20e+05	7,21e+06	8,27e+05
67.5	6,03e+04	2,56e+05	8,48e+06	2,18e+06
90	-2,25e+04	-5,86e+04	-1,03e+06	-2,29e+05

Table 5.5: *Mean values of load on the blades in extreme waves as a function of θ*

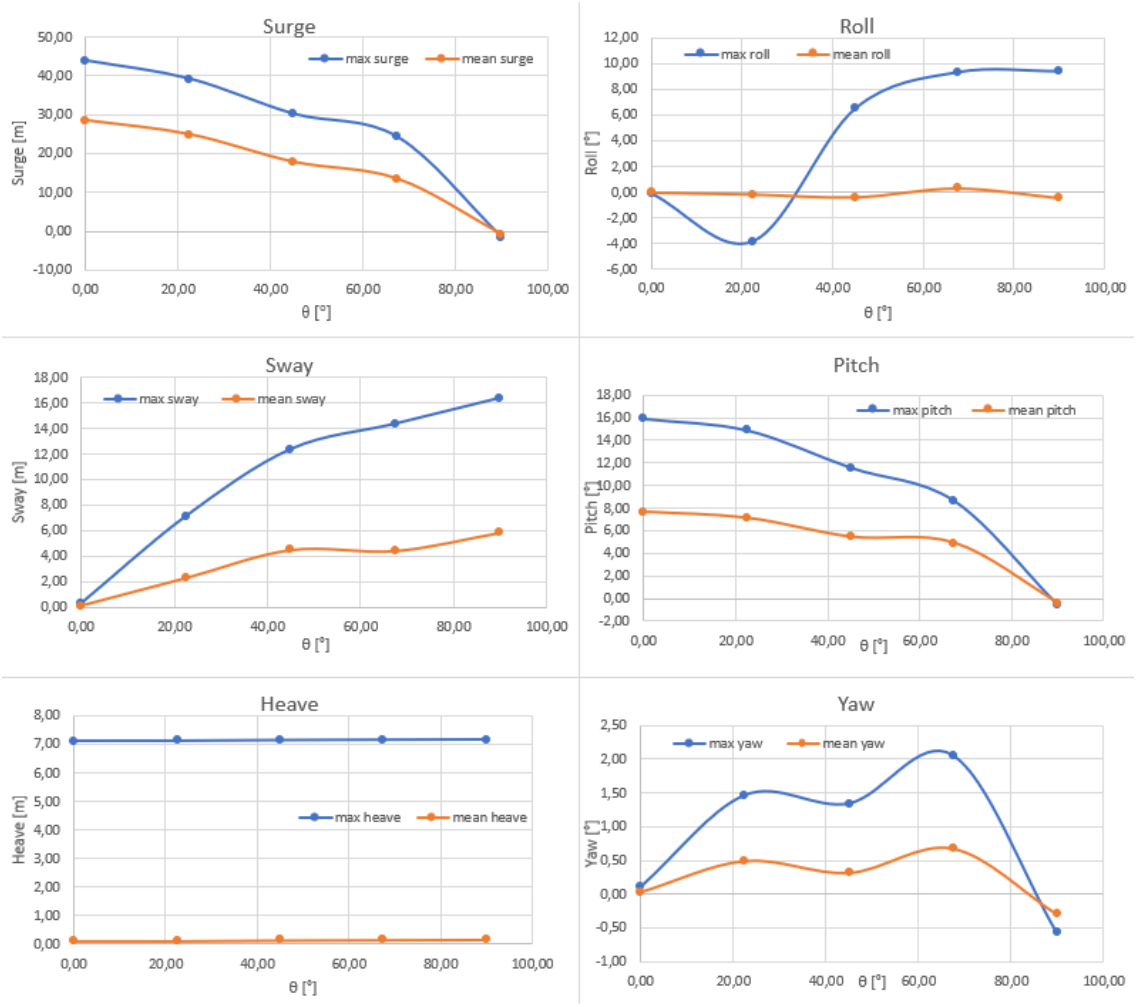


Figure 5.27: *Peak and mean values of floater's motions, extreme wave conditions*

theta	F_τ	F_N	M_f	M_e
0	6,16e+04	3,97e+05	1,31e+07	1,64e+06
22.5	5,32e+04	3,97e+05	1,31e+07	1,58e+06
45	3,60e+04	3,35e+05	1,13e+07	1,27e+06
67.5	9,24e+04	3,82e+05	1,29e+07	3,30e+06
90	-3,45e+04	-8,80e+04	-1,59e+06	-3,61e+05

Table 5.6: *Maximum values of load on the blades in extreme waves as a function of θ*

5.2.4 Misalignment

Often, in the normal operations of a floating system, wind and wave directions can be misaligned: this phenomenon is always present in all atmospheric conditions. In this test case, the incoming wave direction θ is fixed to 0°, while the wind direction γ varies assuming the values 20°, 40°, 60°, that are qualitatively the most probables in the technical report [25], in the case of unstable atmo-

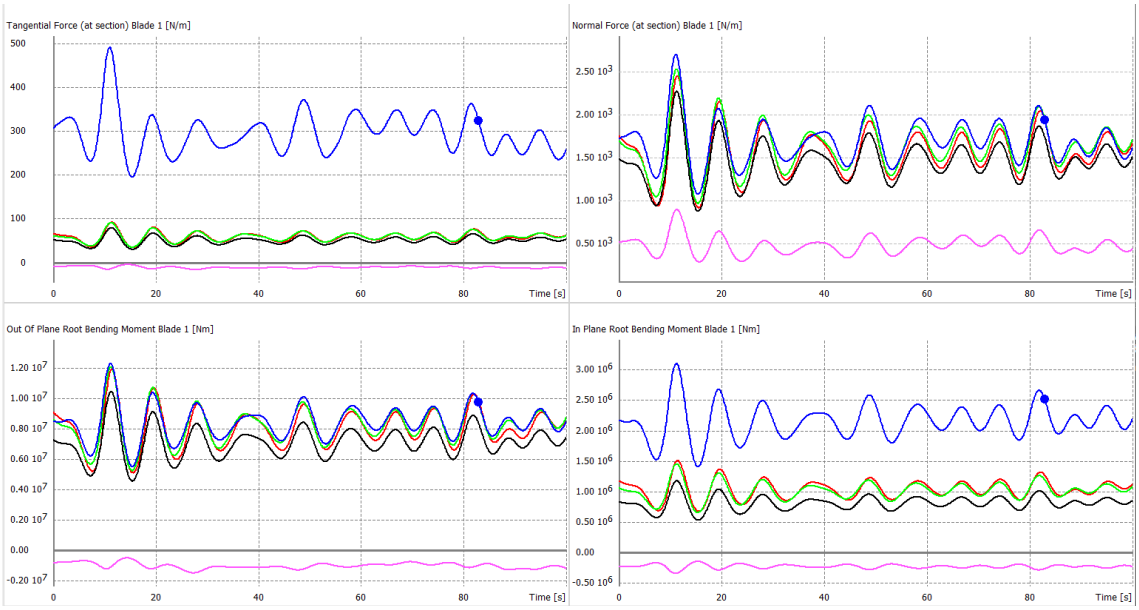


Figure 5.28: *Blade loads in extreme wave conditions, first 100s.*

sphere, modelled with an Hellmann exponent $\alpha = 0.11$. The misalignment angle is defined as

$$misalignment = \gamma - \theta \tag{5.1}$$

Wind speed at hub height U_{hub}, wave height H_s and period T_p are the same of the Normal Operating Conditions Case (Section 5.2.1), while the nacelle yaw angle is always equal to the incoming wind direction γ.

	Misalignment	Colour
	°	
Case 1	20	Red
Case 2	40	Green
Case 3	60	Blue

Table 5.7: Colours Legend for the misalignment case

Kinematics Due to the fact that wind loads are dominant with respect to wave loads, all six degrees of freedom are unlocked in the three cases. Surge decreases as misalignment increases, and conversely sway increases.This regular behaviour cannot be found for the other DOFs. Heave, as in the previous cases, is unaffected by incoming wave/wind direction, and so by the misalignment.

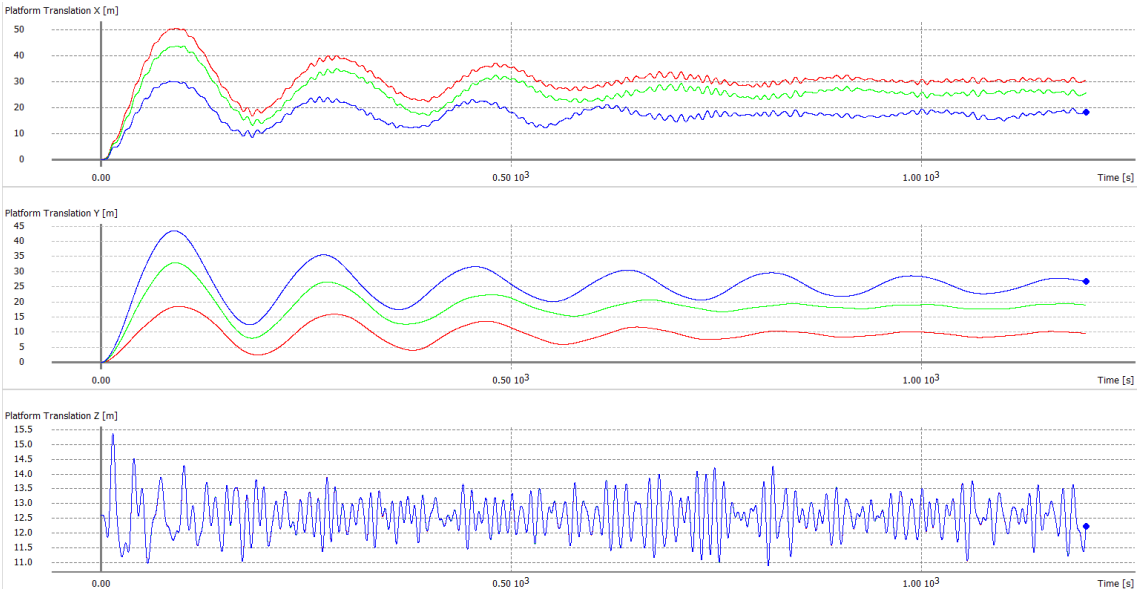


Figure 5.29: *Translations in case of misalignment.*

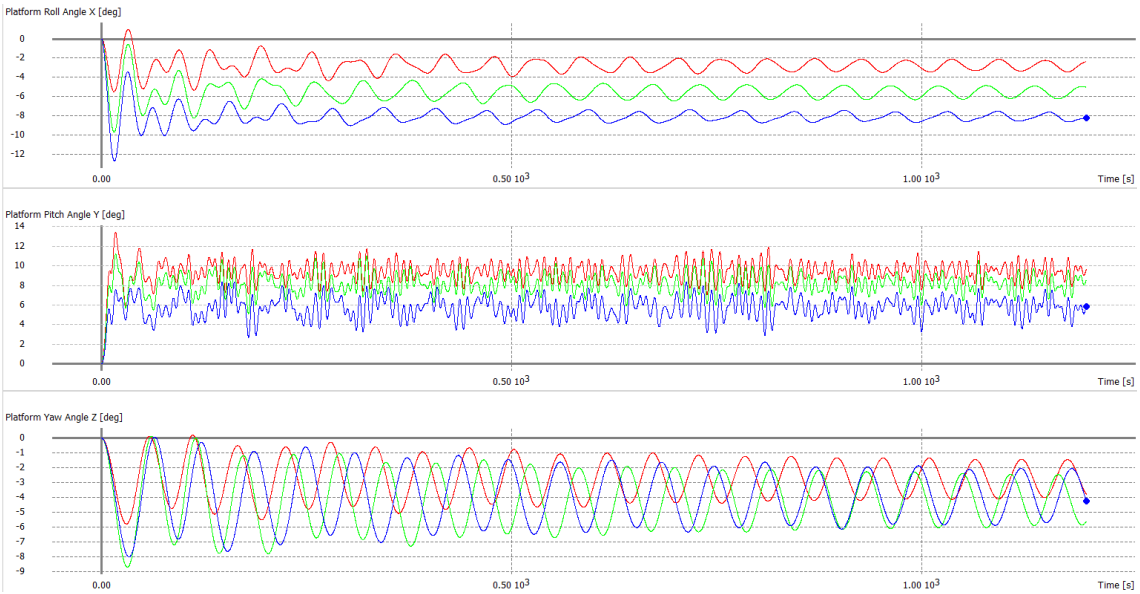


Figure 5.30: *Rotations in case of misalignment.*

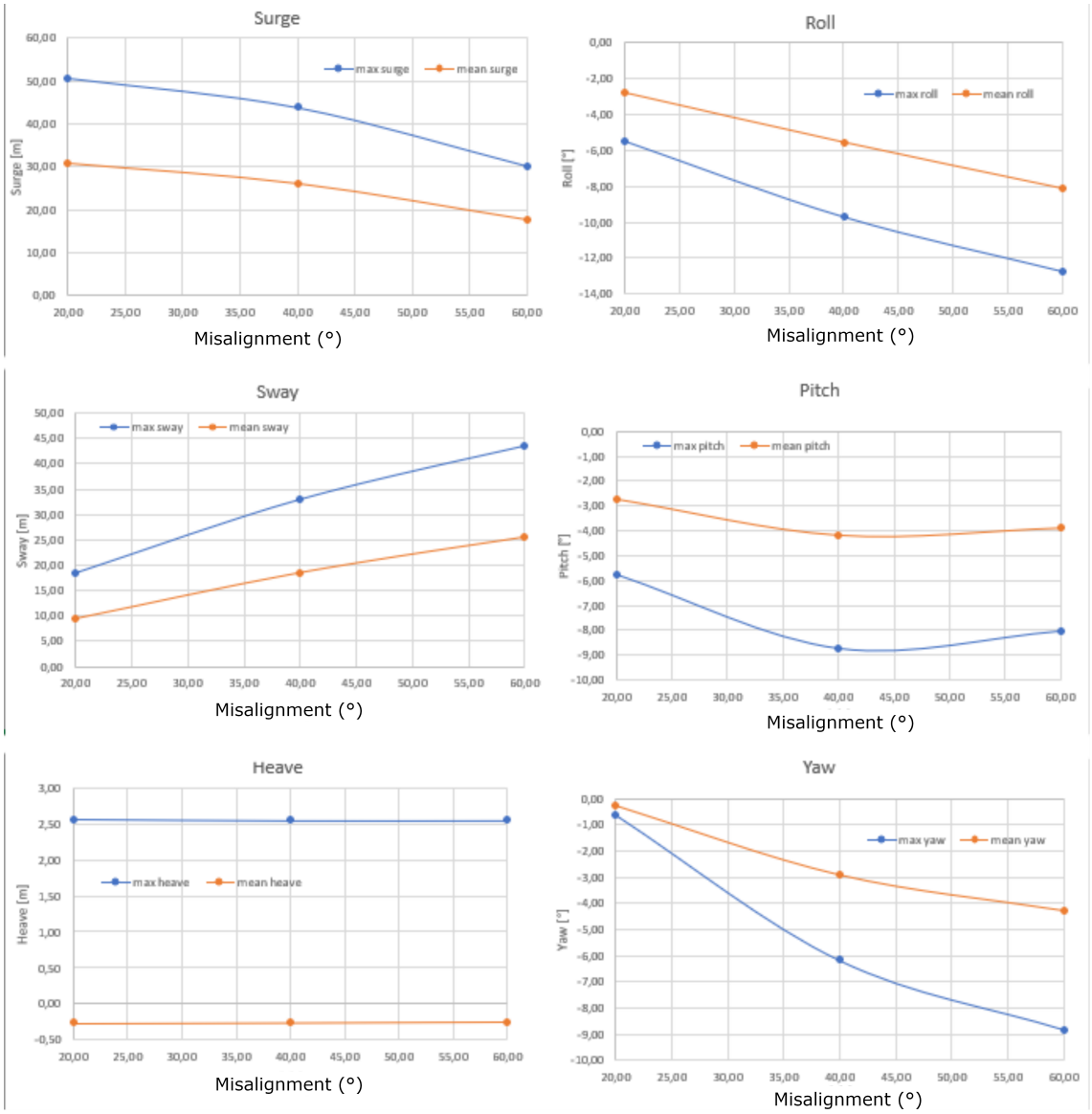


Figure 5.31: *Peak and mean values of floater's motions as a function of the misalignment*

Chapter 6

Final Remarks

6.1 Conclusions

The 6DOFs model has proven to furnish results which are coherent with the expectations. In particular, when wave and wind direction is the one of the positive x axis, sway, yaw and roll are practically locked, and the system behaviour can be accurately described using a 3DOFs model. The variation of the incoming direction of wave and wind unlocks the other degrees of freedom, and for an incoming direction of the loads coincident with the positive y axis, the DOFs locked are surge, pitch and yaw.

The floater's dynamics affects the performances of the wind turbine, in terms of coefficient of power, angle of attack and lift to drag ratio, and in particular cases enhances the probability of stall.

Also blade loads are greatly affected, being the newly-introduced dynamics responsible of cyclic variations much more pronounced than in the onshore case, leading to much more large fatigue stresses.

The coupling between *Floaty* and *QBlade* can be considered as fully accomplished. The two codes communicate well, and the dynamics introduced by the *floater module* has shown to actively influence the behaviour of the wind turbine, in terms of loads on the blades, thrust on the rotor and aerodynamic parameters such as the angle of attack, the drag and the lift coefficients of the blades. Also the development of the wake has shown to be highly dependent on it.

The two codes now form a unique software, having the user the possibility to directly control the floating platform model from *QBlade*, varying the wave to which the platform is subject, the initial position of the system, the mooring parameters and the duration of the simulation. The discretization in terms of direction and time of the wave records can be changed by properly feed the correct files.

The floater results to perform well in the main situations to which the test cases refer, considering that the *non-linear lifting line module* does not actually implement a control logic for the wind turbine.

6.2 Further Investigations

A catenary mooring line model should be implemented: it could characterize better the floater's dynamics, considering that the catenary characteristic is more than proportional. Also coefficients of drag should be revised, getting them through identification from CFD codes or experimental tests.

By revising the source code of *Floaty*, a functionality to change the floating platform can be easily implemented: in this case, all the floater-characterizing parameters, such as the State-Space Matrices, the hydrodynamic stiffness matrix, the floater mass and inertia properties should be written into files, and then given as inputs to the *floater module*. The same is valid for mooring lines' connection points on the floater and at the seabed (in substance, the `polito_floater_data.cpp` source file should be completely removed).

A validation using a reference floater, such as the Floating System for Phase IV of OC3 should be accomplished using validated codes, such as FAST, once the structural dynamics is introduced in the model.

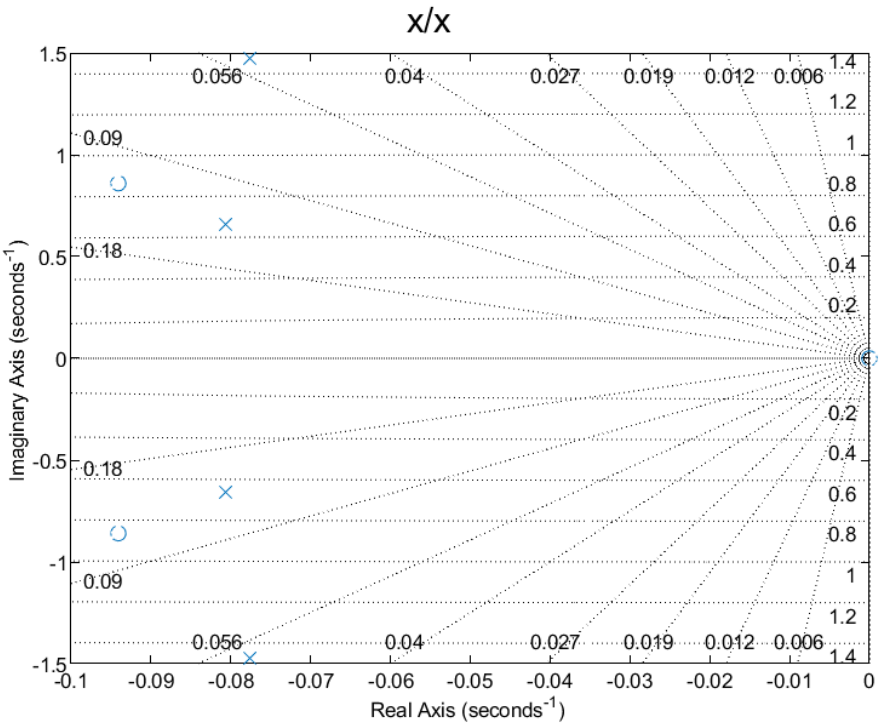
Another code to calculate Froude-Krylov and diffraction forces in real time, as the simulation goes on, should be created and integrated with *Floaty*. The linearity assumptions made in the hydrodynamic model should be gradually replaced by more realistic modelling.

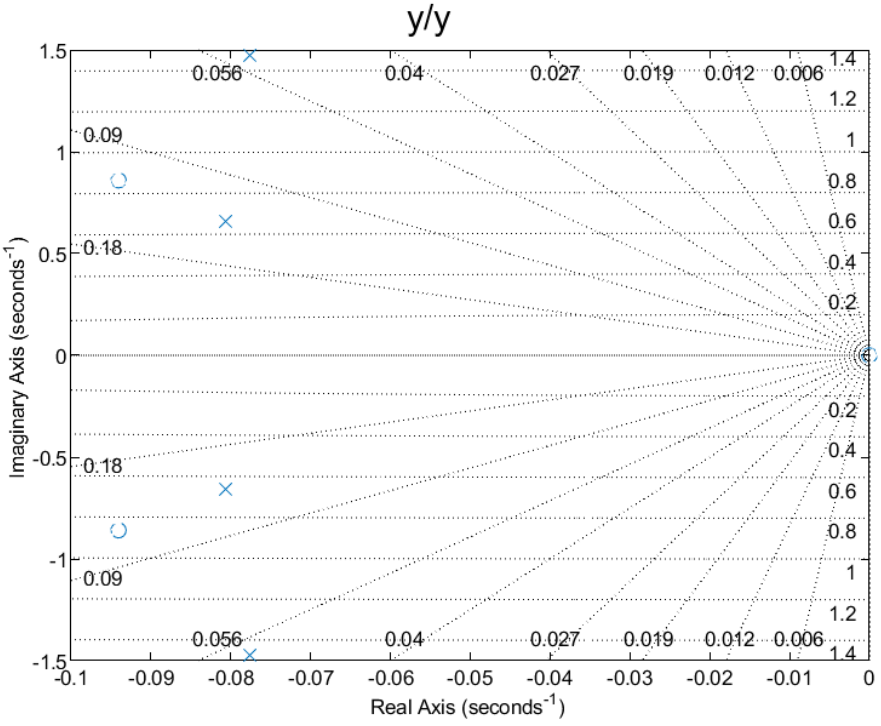
Appendices

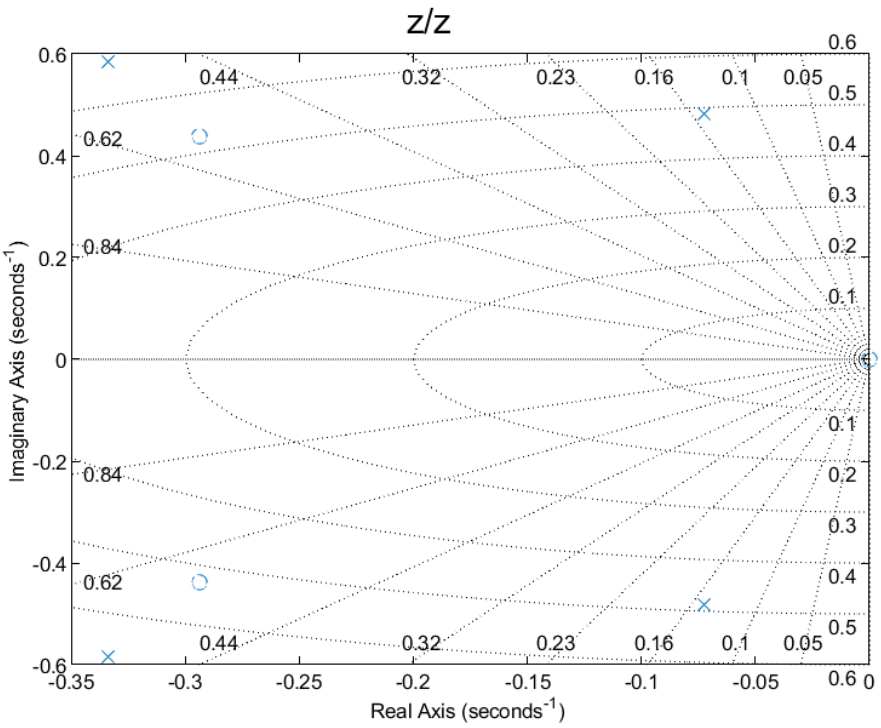
Appendix A

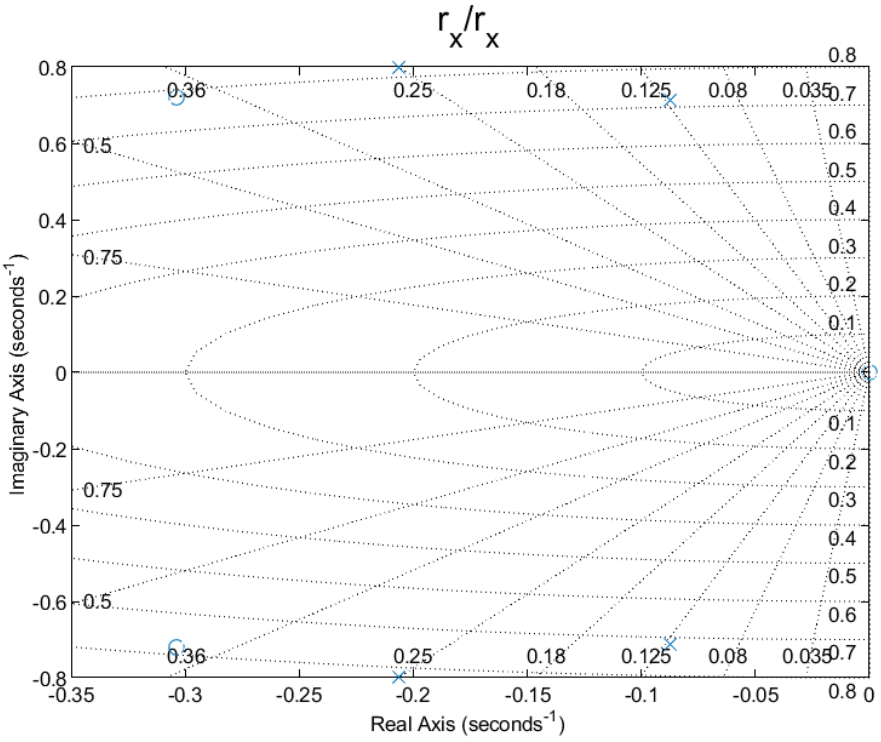
Results of the stability analysis

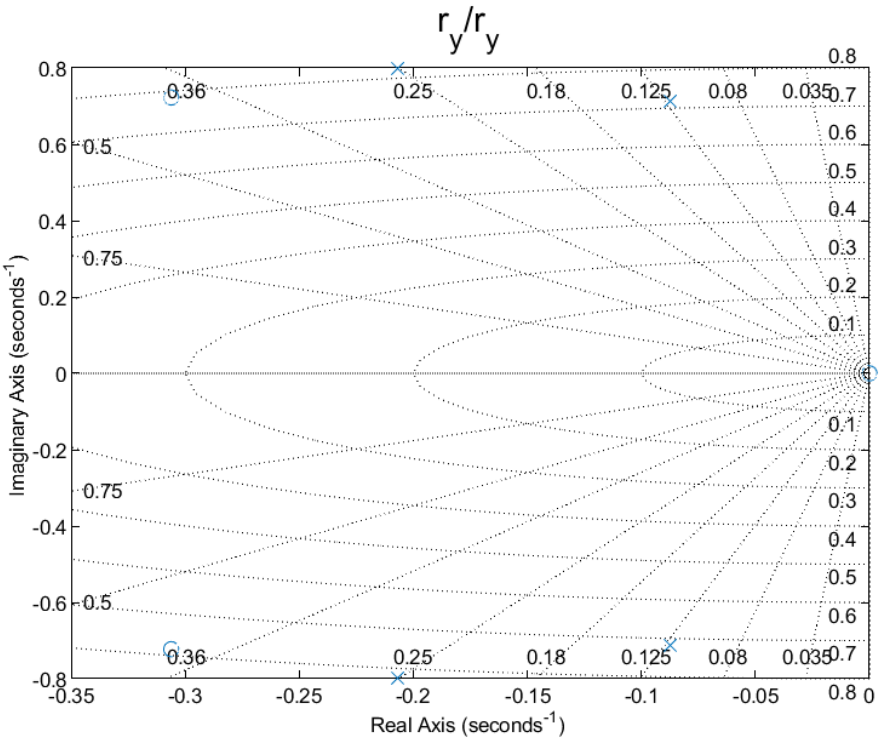
In the following pages are reported the results of the stability analysis. In the figures, the x represents the poles of the transfer function, while the o represents its zeros.

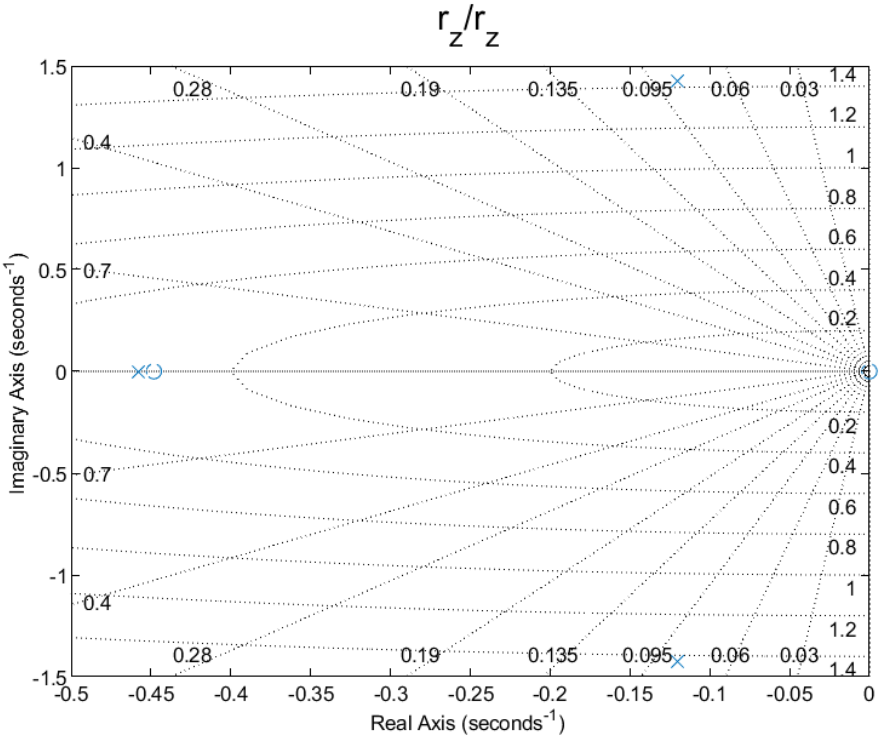

x/x
0.056
0.04
0.027
0.019
0.012
0.006
0.09
0.18
1.5
1
0.5
0
-0.5
-1
-1.5
-0.1
-0.09
-0.08
-0.07
-0.06
-0.05
-0.04
-0.03
-0.02
-0.01
0
1.4
1.2
1
0.8
0.6
0.4
0.2
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})

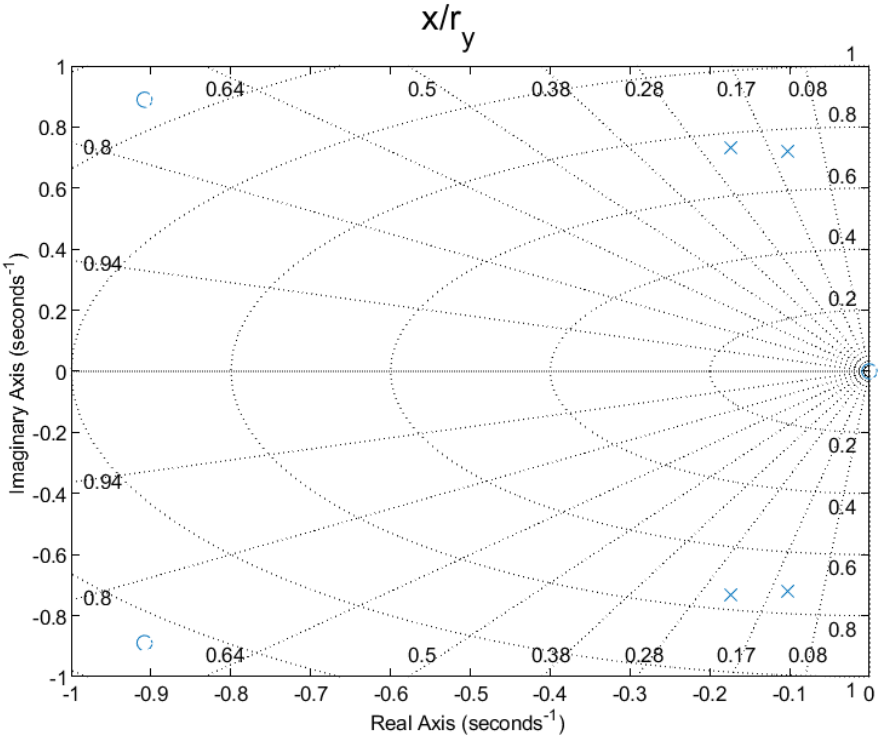

y/y
0.056
0.04
0.027
0.019
0.012
0.006
0.09
0.18
1.5
1
0.5
0
-0.5
-1
-1.5
-0.1
-0.09
-0.08
-0.07
-0.06
-0.05
-0.04
-0.03
-0.02
-0.01
0
1.4
1.2
1
0.8
0.6
0.4
0.2
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})

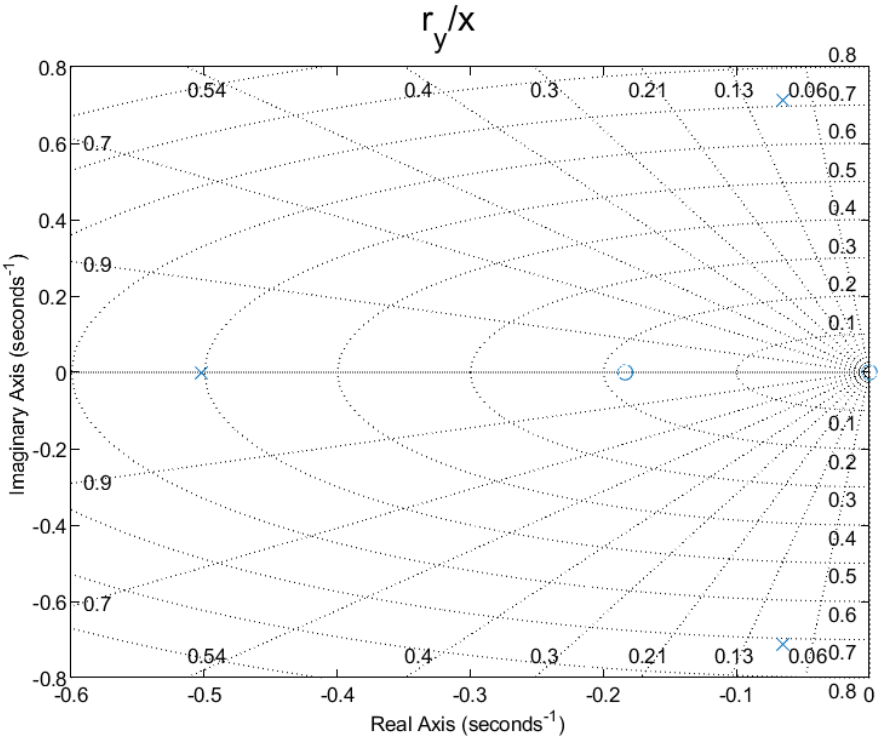

z/z
0.44
0.32
0.23
0.16
0.1
0.05
0.62
0.84
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.2
-0.4
-0.6
-0.35
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
Imaginary Axis (seconds-1)
Real Axis (seconds-1)

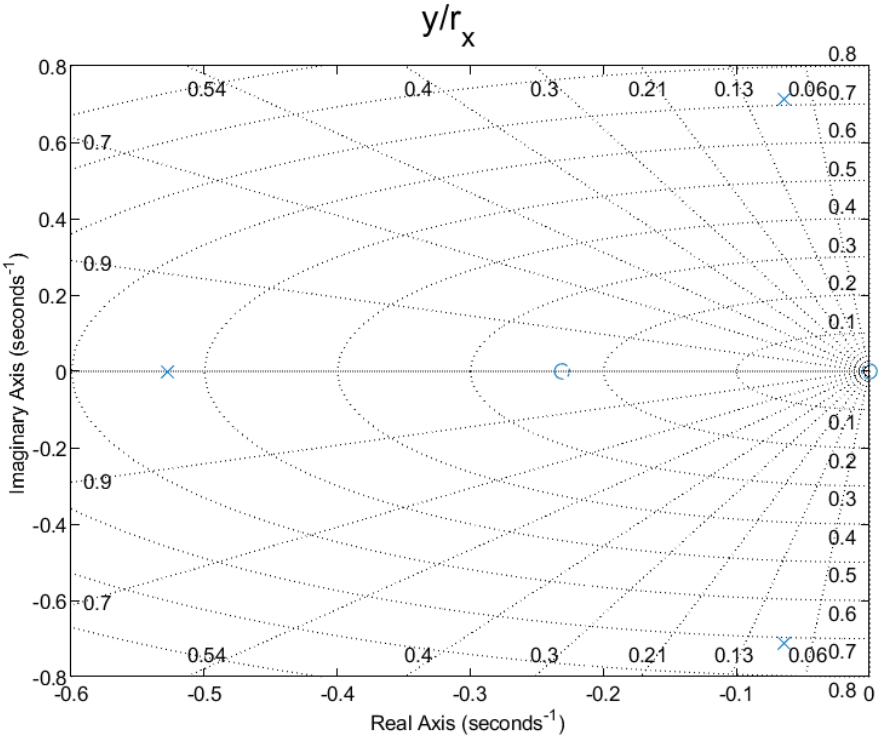

rx/rx
0.36
0.25
0.18
0.125
0.08
0.035
0.5
0.75
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.2
-0.4
-0.6
-0.8
-0.35
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
Imaginary Axis (seconds-1)
Real Axis (seconds-1)

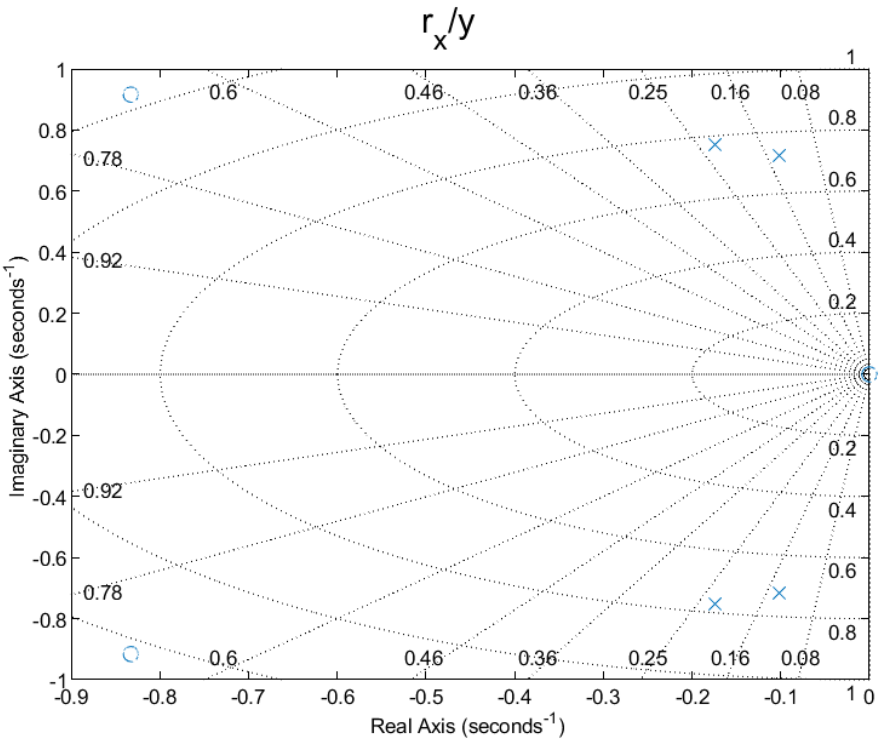

r_y/r_y
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
0.36
0.25
0.18
0.125
0.08
0.035
0.5
0.75
0.7
0.6
0.4
0.3
0.2
0.1
-0.35
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.8
-0.8
-0.6
-0.4
-0.2


r_z/r_z
Imaginary Axis (seconds^{-1})
Real Axis (seconds^{-1})
0.28
0.19
0.135
0.095
0.06
0.03
0.4
0.7
1.4
1.2
1
0.8
0.6
0.4
0.2
-0.5
-0.45
-0.4
-0.35
-0.3
-0.25
-0.2
-0.15
-0.1
-0.05
0
1.5
-1.5
-1


x/r_y
Imaginary Axis (seconds^-1)
Real Axis (seconds^-1)
0.64
0.5
0.38
0.28
0.17
0.08
0.8
0.94


r_y/x
Imaginary Axis (seconds^-1)
Real Axis (seconds^-1)
0.54
0.4
0.3
0.21
0.13
0.06
0.7
0.9


y/r_x
Imaginary Axis (seconds^-1)
Real Axis (seconds^-1)
0.54
0.4
0.3
0.21
0.13
0.06
0.7
0.9

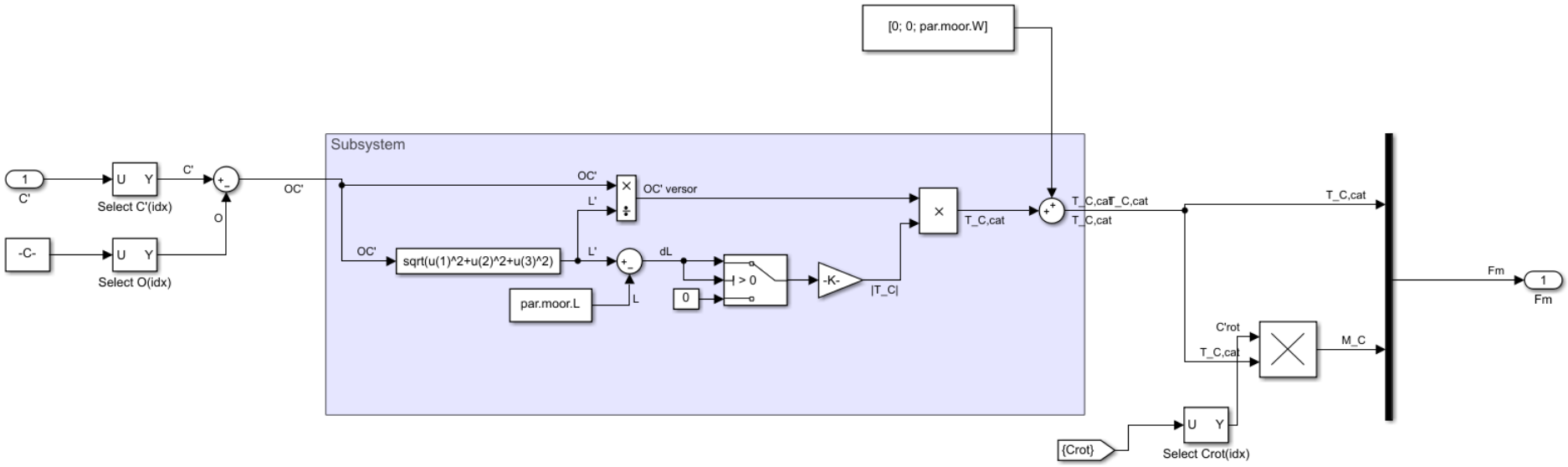

r_x/y
Imaginary Axis (seconds^-1)
Real Axis (seconds^-1)
0.6
0.46
0.36
0.25
0.16
0.08
0.78
0.92

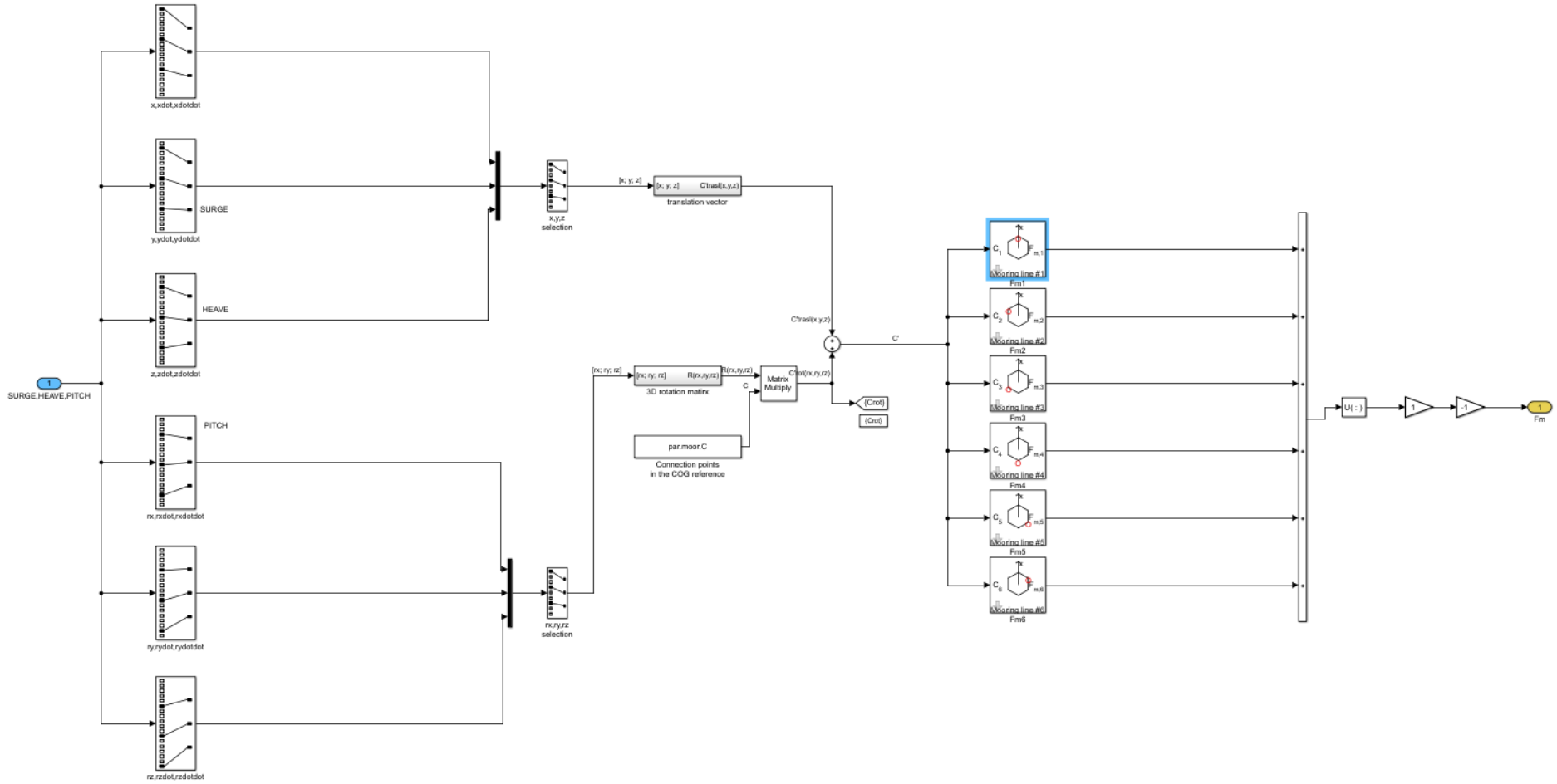
Appendix B

Simulink© Model

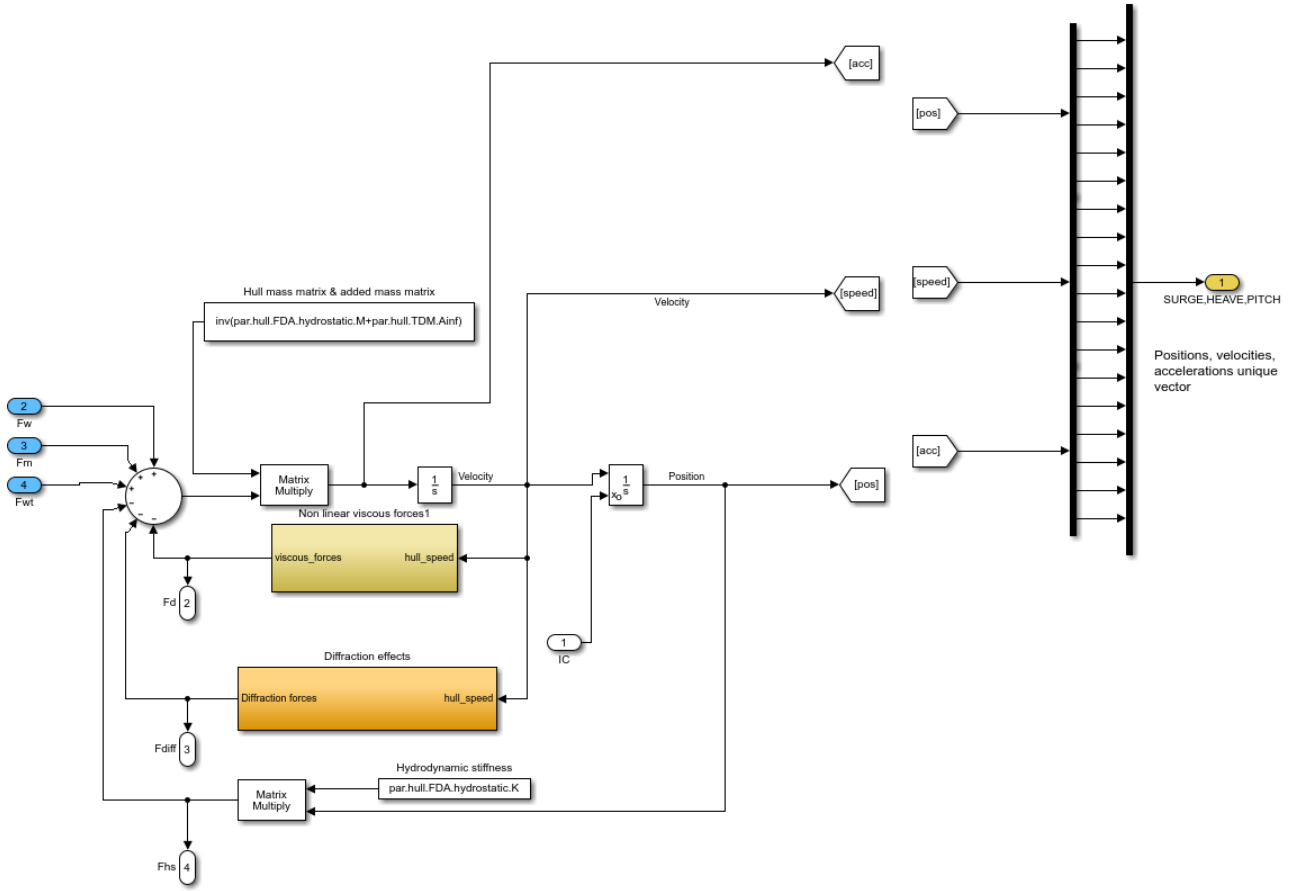
In this Appendix the Simulink models used for the mooring lines and the hull are reported.

B.1 Moorings


[0; 0; par.moor.W]
Subsystem
1
C'
U Y
Select C'(idx)
C'
-C-
U Y
Select O(idx)
O
OC'
OC'
OC' versor
L'
OC'
sqrt(u(1)^2+u(2)^2+u(3)^2)
L'
dL
par.moor.L
L
0
> 0
-K-
|T_C|
T_C,cat
T_C,cat
T_C,cat
T_C,cat
T_C,cat
Fm
1
Fm
C'rot
T_C,cat
M_C
U Y
{Crot}
Select Crot(idx)


1
SURGE,HEAVE,PITCH
x,xdot,xdotdot
SURGE
y,ydot,ydotdot
HEAVE
z,zdot,zdotdot
PITCH
rx,rxdot,rxdotdot
ry,rydot,rydotdot
rz,rzdot,rzdotdot
x,y,z selection
rx,ry,rz selection
[x; y; z]
[x; y; z] C'trasl(x,y,z)
translation vector
[rx; ry; rz]
[rx; ry; rz] R(rx,ry,rz)
3D rotation matirx
R(rx,ry,rz)
C
Matrix Multiply
C'rot(rx,ry,rz)
par.moor.C
Connection points in the COG reference
C'trasl(x,y,z)
C'
{Crot}
{Crot}
Mooring line #1
Fm1
Mooring line #2
Fm2
Mooring line #3
Fm3
Mooring line #4
Fm4
Mooring line #5
Fm5
Mooring line #6
Fm6
U(:)
1
-1
1
Fm

B.2 Hull


[acc]
[pos]
[speed]
1
SURGE,HEAVE,PITCH
Positions, velocities, accelerations unique vector
Hull mass matrix & added mass matrix
inv(par.hull.FDA.hydrostatic.M+par.hull.TDM.Ainf)
Velocity
[speed]
2
Fw
3
Fm
4
Fwt
Matrix Multiply
1/s
Velocity
1/s
Position
[pos]
[acc]
Non linear viscous forces1
viscous_forces
hull_speed
Fd
2
1
IC
Diffraction effects
Diffraction forces
hull_speed
Fdiff
3
Hydrodynamic stiffness
Matrix Multiply
par.hull.FDA.hydrostatic.K
Fhs
4

Appendix C

Coding Details

C.1 Modification of the code produced by Matlab©

`void initialize()` initializes all the variables and blocks needed to run the model, including solver's parameters. First of all, a new structure is declared inside `polito_floater.h` header:

```
...
//simulation parameters
typedef struct {
    real_T dt;
    uint32_T num_wave_el;
    uint32_T num_deg_el;
    real_T moorW;
    real_T moorK;
    real_T moorL;
    real_T moor_pre_tensioning;
} simulation_par;
...
```

It contains, in order, the simulation time step, the number of elements contained in the directions and time lookup table arrays, the mooring lines weight force, stiffness and length. Another structure is added, always in `polito_floater.h` containing all the pointers to the wave force matrix:

```
...
//WAVEFORCE MATRIXES
typedef struct {
    // Expression: FORCEMATRIX(:,:,1)
    //  Referenced by: '<S2>/surge_force'

    real_T *Prelookup_BreakpointsData;

    // Expression: FORCEMATRIX(:,:,1)
    //  Referenced by: '<S2>/surge_force'

    real_T *surge_force_Value;

    // Expression: FORCEMATRIX(:,:,2)
```

```
    //  Referenced by: '<S2>/sway_force'

    real_T *sway_force_Value;

    // Expression: FORCEMATRIX(:,:,3)
    //  Referenced by: '<S2>/heave_force'

    real_T *heave_force_Value;

    // Expression: FORCEMATRIX(:,:,4)
    //  Referenced by: '<S2>/roll_force'

    real_T *roll_force_Value;

    // Expression: FORCEMATRIX(:,:,5)
    //  Referenced by: '<S2>/pitch_force'

    real_T *pitch_force_Value;

    // Expression: FORCEMATRIX(:,:,6)
    //  Referenced by: '<S2>/yaw_force'

    real_T *yaw_force_Value;

    // Expression: -180:10:180
    //  Referenced by: '<S2>/Prelookup1'

    real_T *Prelookup1_BreakpointsData;

} wavef;
...
```

Once this two new structures are added, the declaration and the definition of the initialize function is changed. Here is reported the modification to the source code `polito_floater.cpp` to receive from an external environment all the wave forces. This is done inside the function `void initialize()`.

```
...
// Model initialize function
void polito_floater_ModelClass::initialize(double time[], double
    deg[], double surgeforce[], double swayforce[], double
    heaveforce[], double rollforce[], double pitchforce[], double
    yawforce[])
{

  //initialize pointers to waveforces matrixes
  waveforces.surge_force_Value = surgeforce;
  waveforces.sway_force_Value = swayforce;
  waveforces.heave_force_Value = heaveforce;
  waveforces.roll_force_Value = rollforce;
  waveforces.pitch_force_Value = pitchforce;
  waveforces.yaw_force_Value = yawforce;
  waveforces.Prelookup_BreakpointsData = time;
  waveforces.Prelookup1_BreakpointsData = deg;
```

```
...
```

Code in `void step()` is modified to receive the time step:

```
...
  rtmSetTPtr(getRTM(), &(&polito_floater_M)->Timing.tArray[0]);
  (&polito_floater_M)->Timing.stepSize0 = par.dt;
...
```

Code in `void step()` is modified to receive the parameters related to the prelookup tables:

```
...
// PreLookup: '<S2>/Prelookup'
  rtb_Prelookup_o1 = plook_linx(rtb_InterpolationUsingPreloo_hr,
    waveforces.Prelookup_BreakpointsData, par.num_wave_el - 1U,
    &rtb_InterpolationUsingPreloo_hr);

  // Saturate: '<S2>/Saturation' incorporates:
  //   Inport: '<Root>/wave_dir'

  if (polito_floater_U.wave_dir > 180.0) {
    rtb_Sum = 180.0;
  } else if (polito_floater_U.wave_dir < (-180.0)) {
    rtb_Sum = (-180.0);
  } else {
    rtb_Sum = polito_floater_U.wave_dir;
  }

  // End of Saturate: '<S2>/Saturation'

  // PreLookup: '<S2>/Prelookup1'
  rtb_Prelookup1_o1 = plook_binx(rtb_Sum,
    waveforces.Prelookup1_BreakpointsData, par.num_deg_el - 1U, &
   rtb_Sum);

  // Interpolation_n-D: '<S2>/Interpolation Using Prelookup2'
   incorporates:
  //   Constant: '<S2>/surge_force'

  frac[0] = rtb_InterpolationUsingPreloo_hr;
  frac[1] = rtb_Sum;
  bpIndex[0] = rtb_Prelookup_o1;
  bpIndex[1] = rtb_Prelookup1_o1;
  rtb_InterpolationUsingPrelookup = intrp2d_l_pw(bpIndex, frac,
    waveforces.surge_force_Value, par.num_wave_el);

  // Interpolation_n-D: '<S2>/Interpolation Using Prelookup1'
   incorporates:
  //   Constant: '<S2>/sway_force'

  frac_0[0] = rtb_InterpolationUsingPreloo_hr;
```

```
38   frac_0[1] = rtb_Sum;
     bpIndex_0[0] = rtb_Prelookup_o1;
40   bpIndex_0[1] = rtb_Prelookup1_o1;
     rtb_InterpolationUsingPrelook_f = intrp2d_l_pw(bpIndex_0, frac_0
      ,
42     waveforces.sway_force_Value, par.num_wave_el);

44   // Interpolation_n-D: '<S2>/Interpolation Using Prelookup6'
      incorporates:
     //   Constant: '<S2>/heave_force'
46
     frac_1[0] = rtb_InterpolationUsingPreloo_hr;
48   frac_1[1] = rtb_Sum;
     bpIndex_1[0] = rtb_Prelookup_o1;
50   bpIndex_1[1] = rtb_Prelookup1_o1;
     rtb_InterpolationUsingPreloo_bm = intrp2d_l_pw(bpIndex_1, frac_1
      ,
52     waveforces.heave_force_Value, par.num_wave_el);

54   // Interpolation_n-D: '<S2>/Interpolation Using Prelookup7'
      incorporates:
     //   Constant: '<S2>/roll_force'
56
     frac_2[0] = rtb_InterpolationUsingPreloo_hr;
58   frac_2[1] = rtb_Sum;
     bpIndex_2[0] = rtb_Prelookup_o1;
60   bpIndex_2[1] = rtb_Prelookup1_o1;
     rtb_InterpolationUsingPrelook_h = intrp2d_l_pw(bpIndex_2, frac_2
      ,
62     waveforces.roll_force_Value, par.num_wave_el);

64   // Interpolation_n-D: '<S2>/Interpolation Using Prelookup8'
      incorporates:
     //   Constant: '<S2>/pitch_force'
66
     frac_3[0] = rtb_InterpolationUsingPreloo_hr;
68   frac_3[1] = rtb_Sum;
     bpIndex_3[0] = rtb_Prelookup_o1;
70   bpIndex_3[1] = rtb_Prelookup1_o1;
     rtb_InterpolationUsingPrelook_m = intrp2d_l_pw(bpIndex_3, frac_3
      ,
72     waveforces.pitch_force_Value, par.num_wave_el);

74   // Interpolation_n-D: '<S2>/Interpolation Using Prelookup9'
      incorporates:
     //   Constant: '<S2>/yaw_force'
76
     frac_4[0] = rtb_InterpolationUsingPreloo_hr;
78   frac_4[1] = rtb_Sum;
     bpIndex_4[0] = rtb_Prelookup_o1;
80   bpIndex_4[1] = rtb_Prelookup1_o1;
     rtb_InterpolationUsingPreloo_hr = intrp2d_l_pw(bpIndex_4, frac_4
      ,
82     waveforces.yaw_force_Value, par.num_wave_el);
...
```

Code in `void step()` is modified to receive the parameters related to the mooring lines (the procedure is reported for only one mooring line):

```
...
 //   Fcn: '<S7>/Fcn1'

  rtb_T_C_a = rtb_T_Ccat_b - par.moorL;
...
  // Gain: '<S7>/Gain2'
    rtb_sincos_o2_idx_0 = par.moorK * rtb_T_C_a + par.
   moor_pre_tensioning;
...
  // Sum: '<S7>/Sum4' incorporates:
  //   Constant: '<S7>/Constant6'
  //   Fcn: '<S7>/Fcn1'
  //   Product: '<S7>/Divide'
  //   Product: '<S7>/Divide1'

  rtb_sincos_o1_idx_0 = rtb_sincos_o1_idx_0 / rtb_T_Ccat_b *
   rtb_sincos_o2_idx_0
    + 0.0;
  rtb_sincos_o1_idx_1 = rtb_sincos_o1_idx_1 / rtb_T_Ccat_b *
   rtb_sincos_o2_idx_0
    + 0.0;
  rtb_sincos_o1_idx_2 = rtb_sincos_o1_idx_2 / rtb_T_Ccat_b *
   rtb_sincos_o2_idx_0
    + par.moorW;
...
```

C.2 DLL creation

Once the code is modified, a new *Qt Creator* project is started to create a C++ library. First, all headers and source codes needed to run the simulation are included in the *.pro* file, considering also that to run the Simulink© model in the form of C++ code all Matlab© libraries must be inserted:

```
TARGET = Floaty
TEMPLATE = lib
CONFIG += dll
CONFIG += build64bit
DEFINES += FLOATY_LIBRARY
QT += widgets


INCLUDEPATH += Floaty\MATLAB_Include \
               Floaty\polito_floater_ert_rtw


SOURCES += floaty.cpp\
```

```
        polito_floater_ert_rtw/polito_floater.cpp\
        polito_floater_ert_rtw/polito_floater_data.cpp\
        polito_floater_ert_rtw/rt_nonfinite.cpp\
        polito_floater_ert_rtw/rtGetInf.cpp\
        polito_floater_ert_rtw/rtGetNaN.cpp


HEADERS += floaty.h\
        floaty_global.h\
        MATLAB_Include/solver_zc.h \
        MATLAB_Include/rtw_continuous.h \
        MATLAB_Include/rtw_extmode.h \
        MATLAB_Include/rtw_matlogging.h \
        MATLAB_Include/rtw_solver.h \
        MATLAB_Include/simstruc_types.h \
        MATLAB_Include/solver_zc.h \
        MATLAB_Include/sysran_types.h \
        MATLAB_Include/tmwtypes.h \
        MATLAB_Include/rtw_continous.h \
        MATLAB_Include/rtw_solver.h \
        polito_floater_ert_rtw/polito_floater_private.h \
        polito_floater_ert_rtw/polito_floater_types.h \
        polito_floater_ert_rtw/polito_floater.h \
        polito_floater_ert_rtw/rtwtypes.h \
        polito_floater_ert_rtw/rt_nonfinite.h \
        polito_floater_ert_rtw/rtGetInf.h \
        polito_floater_ert_rtw/rtGetNaN.h
```

The interface of the library is created inside the `floaty.cpp` file. The interface between the code and the external environment is done through a C++ class, declared in the first rows of the source code:

```
#include "floaty.h"
#include "floaty_global.h"
#include "polito_floater_ert_rtw/rtGetInf.h"
#include "polito_floater_ert_rtw/rtGetNaN.h"
#include "polito_floater_ert_rtw/rt_nonfinite.h"
#include "polito_floater_ert_rtw/polito_floater.h"
#include <iostream>
#include <stdlib.h>
#include <QVector>

static polito_floater_ModelClass Floater;
fromModel import;
```

The structure `fromModel` groups all the outputs of the model and is defined as

```
struct fromModel
{
struct kinematics kin;
struct moor_forces Fm;
struct wave_forces Fw;
```

```
struct diff_forces Fdiff;
struct drag_forces Fd;
struct hs_forces Fhs;
};
```

Then, all the functions needed to run the library are declared and defined:

Initialize Function

```
int FirstUse = 1;

extern "C" void __declspec(dllexport) initialize_floater(double
    delta_t, unsigned int deg_el, unsigned int wave_el, double
    moorW, double moorK, double moorL, double preT, double time[],
    double deg[], double surge_table[], double sway_table[], double
     heave_table[], double roll_table[], double pitch_table[],
    double yaw_table[])
{

    if(FirstUse == 1)
    {
        Floater.par.dt = delta_t;
        Floater.par.num_deg_el = deg_el;
        Floater.par.num_wave_el = wave_el;
        Floater.par.moorW = moorW;
        Floater.par.moorK = moorK;
        Floater.par.moorL = moorL;
        Floater.par.moor_pre_tensioning = preT;
        Floater.initialize(time, deg, surge_table, sway_table,
    heave_table, roll_table, pitch_table, yaw_table);
        FirstUse = 0;
    };

    return;
}
```

This function initializes all the arrays containing the wave forces and sets the time step for the simulation. It requires as inputs:

- Simulation time step `delta_t`
- Number of elements in the directions array `deg_el`
- Number of elements contained in the wave time array `wave_el`
- Array containing the time at which each value of the wave forces is computed `time`
- Array containing the directions at which each value of the wave forces is computed `deg`
- Array containing the wave force in the surge direction `surge_table`

- Array containing the wave force in the sway direction `sway_table`
- Array containing the wave force in the heave direction `heave_table`
- Array containing the wave force in the roll direction `roll_table`
- Array containing the wave force in the pitch direction `pitch_table`
- Array containing the wave force in the yaw direction `yaw_table`

The initialize floater function must be called exactly once, before calling the step function.

Step Function

```
extern "C" struct fromModel __declspec(dllexport) platform(double
    wavedir, double Fx, double Fy, double Fz, double Mrx, double
    Mry, double Mrz, double x0, double y0, double z0, double roll0,
     double pitch0, double yaw0){

    //forces exerted by the WT
    Floater.polito_floater_U.Fx = Fx;
    Floater.polito_floater_U.Fy = Fy;
    Floater.polito_floater_U.Fz = Fz;
    //moments exerted by the WT
    Floater.polito_floater_U.Mrx = Mrx;
    Floater.polito_floater_U.Mry = Mry;
    Floater.polito_floater_U.Mrz = Mrz;

    //initial conditions (COG coordinates on the fixed reference
    frame)
    Floater.polito_floater_U.x0 = x0;
    Floater.polito_floater_U.y0 = y0;
    Floater.polito_floater_U.z0 = z0;
    Floater.polito_floater_U.rx0 = roll0;
    Floater.polito_floater_U.ry0 = pitch0;
    Floater.polito_floater_U.rz0 = yaw0;

    //wave direction
    Floater.polito_floater_U.wave_dir = wavedir;

    Floater.step();
```

When this function is called, the simulation goes on by a time step. It requires in input:

- Instantaneous wave direction double `wavedir`
- Force exerted by the wind turbine in the x direction `double Fx`
- Force exerted by the wind turbine in the y direction `double Fy`
- Force exerted by the wind turbine in the z direction `double Fz`
- Moment exerted by the wind turbine in the rx direction `double Mrx`

- Moment exerted by the wind turbine in the ry direction `double Mry`
- Moment exerted by the wind turbine in the rz direction `double Mrz`
- Initial condition for the speed/position integrator in the x direction `double x0`
- Initial condition for the speed/position integrator in the y direction `double y0`
- Initial condition for the speed/position integrator in the z direction `double z0`
- Initial condition for the speed/position integrator in the r_x direction `double rx0`
- Initial condition for the speed/position integrator in the r_y direction `double ry0`
- Initial condition for the speed/position integrator in the r_z direction `double rz0`

Structures to import data from the model

```
//import kinematics variables from the floater model
import.kin.x = Floater.polito_floater_Y.vett[0];
import.kin.y = Floater.polito_floater_Y.vett[1];
import.kin.z = Floater.polito_floater_Y.vett[2];
import.kin.rx = Floater.polito_floater_Y.vett[3];
import.kin.ry = Floater.polito_floater_Y.vett[4];
import.kin.rz = Floater.polito_floater_Y.vett[5];
import.kin.xdot = Floater.polito_floater_Y.vett[6];
import.kin.ydot = Floater.polito_floater_Y.vett[7];
import.kin.zdot = Floater.polito_floater_Y.vett[8];
import.kin.rxdot = Floater.polito_floater_Y.vett[9];
import.kin.rydot = Floater.polito_floater_Y.vett[10];
import.kin.rzdot = Floater.polito_floater_Y.vett[11];
import.kin.xdotdot = Floater.polito_floater_Y.vett[12];
import.kin.ydotdot = Floater.polito_floater_Y.vett[13];
import.kin.zdotdot = Floater.polito_floater_Y.vett[14];
import.kin.rxdotdot = Floater.polito_floater_Y.vett[15];
import.kin.rydotdot = Floater.polito_floater_Y.vett[16];
import.kin.rzdotdot = Floater.polito_floater_Y.vett[17];

//IMPORT MOORING FORCES
import.Fm.Fx = Floater.polito_floater_Y.Fm[0];
import.Fm.Fy = Floater.polito_floater_Y.Fm[1];
import.Fm.Fz = Floater.polito_floater_Y.Fm[2];
import.Fm.Mx = Floater.polito_floater_Y.Fm[3];
import.Fm.My = Floater.polito_floater_Y.Fm[4];
import.Fm.Mz = Floater.polito_floater_Y.Fm[5];

//IMPORT WAVE FORCES
import.Fw.Fx = Floater.polito_floater_Y.Fw[0];
import.Fw.Fy = Floater.polito_floater_Y.Fw[1];
import.Fw.Fz = Floater.polito_floater_Y.Fw[2];
```

```
    import.Fw.Mx = Floater.polito_floater_Y.Fw[3];
    import.Fw.My = Floater.polito_floater_Y.Fw[4];
    import.Fw.Mz = Floater.polito_floater_Y.Fw[5];

    //IMPORT DRAG FORCES
    import.Fd.Fx = Floater.polito_floater_Y.Fd[0];
    import.Fd.Fy = Floater.polito_floater_Y.Fd[1];
    import.Fd.Fz = Floater.polito_floater_Y.Fd[2];
    import.Fd.Mx = Floater.polito_floater_Y.Fd[3];
    import.Fd.My = Floater.polito_floater_Y.Fd[4];
    import.Fd.Mz = Floater.polito_floater_Y.Fd[5];

    //IMPORT DIFFRACTION FORCES
    import.Fdiff.Fx = Floater.polito_floater_Y.Fdiff[0];
    import.Fdiff.Fy = Floater.polito_floater_Y.Fdiff[1];
    import.Fdiff.Fz = Floater.polito_floater_Y.Fdiff[2];
    import.Fdiff.Mx = Floater.polito_floater_Y.Fdiff[3];
    import.Fdiff.My = Floater.polito_floater_Y.Fdiff[4];
    import.Fdiff.Mz = Floater.polito_floater_Y.Fdiff[5];

    //IMPORT HYDROSTATIC FORCES
    import.Fhs.Fx = Floater.polito_floater_Y.Fhs[0];
    import.Fhs.Fy = Floater.polito_floater_Y.Fhs[1];
    import.Fhs.Fz = Floater.polito_floater_Y.Fhs[2];
    import.Fhs.Mx = Floater.polito_floater_Y.Fhs[3];
    import.Fhs.My = Floater.polito_floater_Y.Fhs[4];
    import.Fhs.Mz = Floater.polito_floater_Y.Fhs[5];

    return import;
}
```

Close Function

This function closes the floater model and calls the destructor, it must be called at the end of the simulation.

```
extern "C" void __declspec(dllexport) close_floater()
{
    Floater.terminate();
    Floater.~polito_floater_ModelClass();
    FirstUse = 1;
    return;
}
```

Naturally, all the function and variables declarations are reported in the header file `floaty.h`. Once all the modifications are done, the library is ready to be implemented in an external software that uses it.

C.3 Interfacing with *QBlade*

C.3.1 Import the waves

To import the waves, a new member function is added to the class `QLLTSimulation`. It basically receives the directory chosen by the user in the GUI and loads all the data, stored in form of `.dat` files.

```
//POLITO FLOATER
void QLLTSimulation::onLoadFloaterParameters()
{
unsigned int cont = 0;

//********************IMPORT SIMULATION PARAMETERS
    ********************//
QString line;
QFile import_wave;
unsigned int matrixsize = num_wave*num_deg;

//***WAVEFORCE ARRAYS DEFINITIONS***//

surge_force = new double[matrixsize];
sway_force = new double[matrixsize];
heave_force = new double[matrixsize];
roll_force = new double[matrixsize];
pitch_force = new double[matrixsize];
yaw_force = new double[matrixsize];

//***LOADING DIRECTIONS AND TIME ARRAY FOR PRELOOKUP TABLE***//
deg = new double[num_deg];
time = new double[num_wave];

//***LOADING THE DIRECTIONS OF THE PRELOOKUP TABLE***//
cont = 0;
import_wave.setFileName(wave_dir + "/deg.dat");

if(!import_wave.open(QIODevice::ReadOnly))
{
    AbortSimulation();
}

while(cont < num_deg)
{
    line = import_wave.readLine();
    deg[cont] = line.toDouble();
    cont ++;
}

fputs("Directions array acquired correctly\n", stdout);
cont = 0;
import_wave.close();

//***IMPORT THE WAVE FORCE TIMESET***//

import_wave.setFileName(wave_dir + "/time.dat");
```

```
47 if(!import_wave.open(QIODevice::ReadOnly))
   {
49    AbortSimulation();
   }
51
   while(cont < num_wave)
53 {
      line = import_wave.readLine();
55    time[cont] = line.toDouble();
      cont ++;
57 }

59 fputs("Time array acquired correctly\n", stdout);
   cont = 0;
61 import_wave.close();

63 //********************IMPORT WAVE FORCES********************//
   //***SURGE FORCE***//
65 if(!import_wave.open(QIODevice::ReadOnly))
   {
67    AbortSimulation();
   }
69
   while(cont < matrixsize)
71 {
      line = import_wave.readLine();
73    surge_force[cont] = line.toDouble();
      cont ++;
75 }

77 fputs("Surge forces array acquired correctly\n", stdout);
   cont = 0;
79 import_wave.close();


81
   //***SWAY FORCE***//
83 import_wave.setFileName(wave_dir + "/sway.dat");
   if(!import_wave.open(QIODevice::ReadOnly))
85 {
      AbortSimulation();
87 }

89 while(cont < matrixsize)
   {
91    line = import_wave.readLine();
      sway_force[cont] = line.toDouble();
93    cont ++;
   }
95
   fputs("Sway forces array acquired correctly\n", stdout);
97 cont = 0;
   import_wave.close();
99
   //***HEAVE FORCE***//
101 import_wave.setFileName(wave_dir + "/heave.dat");
    if(!import_wave.open(QIODevice::ReadOnly))
```

```
{
    AbortSimulation();
}

while(cont < matrixsize)
{
    line = import_wave.readLine();
    heave_force[cont] = line.toDouble();
    cont ++;
}

fputs("Heave forces array acquired correctly\n", stdout);
cont = 0;
import_wave.close();

//***ROLL FORCES***//
import_wave.setFileName(wave_dir + "/roll.dat");
if(!import_wave.open(QIODevice::ReadOnly))
{
    AbortSimulation();
}

while(cont < matrixsize)
{
    line = import_wave.readLine();
    roll_force[cont] = line.toDouble();
    cont ++;
}

fputs("Roll forces array acquired correctly\n", stdout);
cont = 0;
import_wave.close();


//***PITCH FORCES***//
import_wave.setFileName(wave_dir+ "/pitch.dat");
if(!import_wave.open(QIODevice::ReadOnly))
{
    AbortSimulation();
}

while(cont < matrixsize)
{
    line = import_wave.readLine();
    pitch_force[cont] = line.toDouble();
    cont ++;
}

fputs("Pitch forces array acquired correctly\n", stdout);
cont = 0;
import_wave.close();

//***YAW FORCES***//
import_wave.setFileName(wave_dir + "/yaw.dat");
if(!import_wave.open(QIODevice::ReadOnly))
{
```

```
159     AbortSimulation();
}
161
while(cont < matrixsize)
163 {
    line = import_wave.readLine();
165     yaw_force[cont] = line.toDouble();
    cont ++;
167 }

169 fputs("Yaw forces array acquired correctly\n", stdout);
cont = 0;
171 import_wave.close();

173 initialize_floater(getTimeStep(), num_deg, num_wave, m_moorW,
    m_moorK, m_moorL, preT, time, deg, surge_force, sway_force,
    heave_force, roll_force, pitch_force, yaw_force);

175 import_wave.close();
}
```

C.3.2 Call the **platform** function

It is added inside the member function `calcHAWTresults`

```
...
2
    //****load waves if floater is active****//
4     if(active_floater & !m_bContinue){
    onLoadFloaterParameters();
6     }

8     for (; m_currentTimeStep <= m_numTimesteps; ++
    m_currentTimeStep) {

10         emit updateProgress(m_currentTimeStep);

12         qDebug() << "start calculation for timestep:" <<
    m_currentTimeStep << "Input Type"<<m_windInputType<< "Azi Pos
    Blade 1"<<m_currentAzimuthalPosition;

14         SetBoundaryConditions();

16         UpdateRotorGeometry();

18         if (m_currentTimeStep == m_Nth_WakeStep){
               //start first wake timestep
20             AddFirstWake();
               KuttaCondition();
22         }
           else if(m_currentTimeStep > m_Nth_WakeStep && (
    m_currentTimeStep)%m_Nth_WakeStep == 0){
24             //start sucessive wake timesteps
```

```
            TruncateWake();
            LumpWake();                             m_t_overhead +=
    timer.restart();
            ConvectWake();                          m_t_induction+=
    timer.restart();
            AddNewWake();
            KuttaCondition();
        }

        //iteration for wing circulation distribution
                                                    m_t_overhead+=
    timer.restart();
        GammaBoundFixedPointIteration();            m_t_induction+=
    timer.restart();

        if(active_floater){

/calculation of moments exerted by the wind turbine on the COG of
    the floater.
mx.append(-ThrustLSAy.last()*fabs(m_ThrustActingPoint.data()->z));
my.append(ThrustLSAz.last()*fabs(m_ThrustActingPoint.data()->x) +
    ThrustLSAx.last()*fabs(m_ThrustActingPoint.data()->z));
mz.append(-ThrustLSAy.last()*fabs(m_ThrustActingPoint.data()->x));

floater =  platform(wave_direction, m_ThrustX.last(), m_ThrustY.
    last(), m_ThrustZ.last(), mx.last(), my.last(), mz.last(),
m_PlatformTranslation.x, m_PlatformTranslation.y, swl_cog_floaty,
    m_PlatformRollAngleX*PI/180, m_PlatformPitchAngleY*PI/180,
    m_PlatformYawAngleZ*PI/180);

        swl_cog_floaty = floater.kin.z;
        swl_bt = swl_cog_floaty + cog_b;
        m_PlatformTranslation.x = floater.kin.x;
        m_PlatformTranslation.y = floater.kin.y;
        m_PlatformTranslation.z = swl_bt;
        m_PlatformYawAngleZ = floater.kin.rz*180/PI;
        m_PlatformRollAngleX = floater.kin.rx*180/PI;
        m_PlatformPitchAngleY = floater.kin.ry*180/PI;

...
```

C.3.3 Store the results inside arrays

Once the results are acquired from the structure `floater`, they are stored inside vectors that are all members of the class `QLLTSimulation`. They are all declared inside the header `QLLTSimulation.h`.

```
...

        //******START IMPORTING THE RESULTS TO QBLADE******//
        //KINEMATICS
        //SPEED
        xdot.append(floater.kin.xdot);
```

```
        ydot.append(floater.kin.ydot);
        zdot.append(floater.kin.zdot);
        rxdot.append(floater.kin.rxdot*180/PI);
        rydot.append(floater.kin.rydot*180/PI);
        rzdot.append(floater.kin.rzdot*180/PI);
        //ACCELERATIONS
        xdotdot.append(floater.kin.xdotdot);
        ydotdot.append(floater.kin.ydotdot);
        zdotdot.append(floater.kin.zdotdot);
        rxdotdot.append(floater.kin.rxdotdot*180/PI);
        rydotdot.append(floater.kin.rydotdot*180/PI);
        rzdotdot.append(floater.kin.rzdotdot*180/PI);
        //DRAG_FORCES
        drag_Fx.append(floater.Fd.Fx);
        drag_Fy.append(floater.Fd.Fy);
        drag_Fz.append(floater.Fd.Fz);
        drag_Mrx.append(floater.Fd.Mx);
        drag_Mry.append(floater.Fd.My);
        drag_Mrz.append(floater.Fd.Mz);
        //MOORING FORCES
        moor_Fx.append(floater.Fm.Fx);
        moor_Fy.append(floater.Fm.Fy);
        moor_Fz.append(floater.Fm.Fz);
        moor_Mrx.append(floater.Fm.Mx);
        moor_Mry.append(floater.Fm.My);
        moor_Mrz.append(floater.Fm.Mz);
        //WAVE FORCES
        wave_Fx.append(floater.Fw.Fx);
        wave_Fy.append(floater.Fw.Fy);
        wave_Fz.append(floater.Fw.Fz);
        wave_Mrx.append(floater.Fw.Mx);
        wave_Mry.append(floater.Fw.My);
        wave_Mrz.append(floater.Fw.Mz);
        //RADIATION FORCES
        rad_Fx.append(floater.Fdiff.Fx);
        rad_Fy.append(floater.Fdiff.Fy);
        rad_Fz.append(floater.Fdiff.Fz);
        rad_Mrx.append(floater.Fdiff.Mx);
        rad_Mry.append(floater.Fdiff.My);
        rad_Mrz.append(floater.Fdiff.Mz);
        //HYDROSTATIC FORCES
        HS_Fx.append(floater.Fhs.Fx);
        HS_Fy.append(floater.Fhs.Fy);
        HS_Fz.append(floater.Fhs.Fz);
        HS_Mrx.append(floater.Fhs.Mx);
        HS_Mry.append(floater.Fhs.My);
        HS_Mrz.append(floater.Fhs.Mz);

...
```

C.3.4 Send the arrays to the Plot Creator

To permit the user to plot the variables coming from the *Floate Module*, the member function `PrepareOutputVectors` inside the class `QLLTSimulation` is edited:

```
...

if(active_floater)
{
//KINEMATICS
m_availableVariables.append("Platform Pitch speed [deg/s]");
m_results.append(&rydot);
m_availableVariables.append("Platform Roll speed [deg/s]");
m_results.append(&rxdot);
m_availableVariables.append("Platform Yaw speed [deg/s]");
m_results.append(&rzdot);
m_availableVariables.append("Platform X speed [m/s]");
m_results.append(&xdot);
m_availableVariables.append("Platform Y speed [m/s]");
m_results.append(&ydot);
m_availableVariables.append("Platform Z speed [m/s]");
m_results.append(&zdot);
m_availableVariables.append("Platform Pitch acceleration [deg/s^2]
    ");
m_results.append(&rydotdot);
m_availableVariables.append("Platform Roll acceleration [deg/s^2]"
    );
m_results.append(&rxdotdot);
m_availableVariables.append("Platform Yaw acceleration [deg/s^2]")
    ;
m_results.append(&rzdotdot);
m_availableVariables.append("Platform X acceleration [deg/s]");
m_results.append(&xdotdot);
m_availableVariables.append("Platform Y acceleration [deg/s]");
m_results.append(&ydotdot);
m_availableVariables.append("Platform Z acceleration [deg/s]");
m_results.append(&zdotdot);
//DRAG FORCES
m_availableVariables.append("HydroDrag X force [N]");
m_results.append(&drag_Fx);
m_availableVariables.append("HydroDrag Y force [N]");
m_results.append(&drag_Fy);
m_availableVariables.append("HydroDrag Z force [N]");
m_results.append(&drag_Fz);
m_availableVariables.append("HydroDrag roll moment [Nm]");
m_results.append(&drag_Mrx);
m_availableVariables.append("HydroDrag pitch moment [Nm]");
m_results.append(&drag_Mry);
m_availableVariables.append("HydroDrag yaw moment [Nm]");
 m_results.append(&drag_Mrz);
//MOORING FORCES
m_availableVariables.append("Moorings X force [N]");
m_results.append(&moor_Fx);m_availableVariables.append("Moorings Y
     force [N]");
m_results.append(&moor_Fy);
```

```
47 m_availableVariables.append("Moorings Z force [N]");
   m_results.append(&moor_Fz);
49 m_availableVariables.append("Moorings roll moment [Nm]");
   m_results.append(&moor_Mrx);
51 m_availableVariables.append("Moorings pitch moment [Nm]");
   m_results.append(&moor_Mry);
53 m_availableVariables.append("Moorings yaw moment [Nm]");
   m_results.append(&moor_Mrz);
55 //HYDROSTATIC FORCES
   m_availableVariables.append("Hydrostatic X force [N]");
57 m_results.append(&HS_Fx);
   m_availableVariables.append("Hydrostatic Y force [N]");
59 m_results.append(&HS_Fy);
   m_availableVariables.append("Hydrostatic Z force [N]");
61 m_results.append(&HS_Fz);
   m_availableVariables.append("Hydrostatic roll moment [Nm]");
63 m_results.append(&HS_Mrx);
   m_availableVariables.append("Hydrostatic pitch moment [Nm]");
65 m_results.append(&HS_Mry);
   m_availableVariables.append("Hydrostatic yaw moment [Nm]");
67 m_results.append(&HS_Mrz);
   //RADIATION FORCES
69 m_availableVariables.append("Radiation X force [N]");
   m_results.append(&rad_Fx);
71 m_availableVariables.append("Radiation Y force [N]");
   m_results.append(&rad_Fy);
73 m_availableVariables.append("Radiation Z force [N]");
   m_results.append(&rad_Fz);
75 m_availableVariables.append("Radiation roll moment [Nm]");
   m_results.append(&rad_Mrx);
77 m_availableVariables.append("Radiation pitch moment [Nm]");
   m_results.append(&rad_Mry);
79 m_availableVariables.append("Radiation yaw moment [Nm]");
   m_results.append(&rad_Mrz);
81 //WAVE FORCES
   m_availableVariables.append("Wave X force [N]");
83 m_results.append(&wave_Fx);
   m_availableVariables.append("Wave Y force [N]");
85 m_results.append(&wave_Fy);
   m_availableVariables.append("Wave Z force [N]");
87 m_results.append(&wave_Fz);
   m_availableVariables.append("Wave roll moment [Nm]");
89 m_results.append(&wave_Mrx);
   m_availableVariables.append("Wave pitch moment [Nm]");
91 m_results.append(&wave_Mry);
   m_availableVariables.append("Wave yaw moment [Nm]");
93 m_results.append(&wave_Mrz);
   //thrust in the LSA FRAME
95 m_availableVariables.append("Thrust LSA X [N]");
   m_results.append(&ThrustLSAx);
97 m_availableVariables.append("Thrust LSA Y [N]");
   m_results.append(&ThrustLSAy);
99 m_availableVariables.append("Thrust LSA Z [N]");
   m_results.append(&ThrustLSAz);
101 //moment in the COG
   m_availableVariables.append("Moment LSA X [Nm]");
```

```
m_results.append(&mx);
m_availableVariables.append("Moment LSA Y [Nm]");
m_results.append(&my);
m_availableVariables.append("Moment LSA Z [Nm]");
m_results.append(&mz);
        };

...
```

C.4 Export Floaty results to the .wpa project file

All the output variables from the *floater module* are stored, at each time step, inside the simulation output file, including in the `serialize` member function the following code:

```
...
//POLITO FLOATER
//KINEMATICS
g_serializer.readOrWriteDoubleVector1D(&rydot);
g_serializer.readOrWriteDoubleVector1D(&rxdot);
g_serializer.readOrWriteDoubleVector1D(&rzdot);
g_serializer.readOrWriteDoubleVector1D(&xdot);
g_serializer.readOrWriteDoubleVector1D(&ydot);
g_serializer.readOrWriteDoubleVector1D(&zdot);
g_serializer.readOrWriteDoubleVector1D(&rydotdot);
g_serializer.readOrWriteDoubleVector1D(&rxdotdot);
g_serializer.readOrWriteDoubleVector1D(&rzdotdot);
g_serializer.readOrWriteDoubleVector1D(&xdotdot);
g_serializer.readOrWriteDoubleVector1D(&ydotdot);
g_serializer.readOrWriteDoubleVector1D(&zdotdot);
//DRAG FORCES
g_serializer.readOrWriteDoubleVector1D(&drag_Fx);
g_serializer.readOrWriteDoubleVector1D(&drag_Fy);
g_serializer.readOrWriteDoubleVector1D(&drag_Fz);
g_serializer.readOrWriteDoubleVector1D(&drag_Mrx);
g_serializer.readOrWriteDoubleVector1D(&drag_Mry);
g_serializer.readOrWriteDoubleVector1D(&drag_Mrz);
//MOORING FORCES
g_serializer.readOrWriteDoubleVector1D(&moor_Fx);
g_serializer.readOrWriteDoubleVector1D(&moor_Fy);
g_serializer.readOrWriteDoubleVector1D(&moor_Fz);
g_serializer.readOrWriteDoubleVector1D(&moor_Mrx);
g_serializer.readOrWriteDoubleVector1D(&moor_Mry);
g_serializer.readOrWriteDoubleVector1D(&moor_Mrz);
//HYDROSTATIC FORCES
g_serializer.readOrWriteDoubleVector1D(&HS_Fx);
g_serializer.readOrWriteDoubleVector1D(&HS_Fy);
g_serializer.readOrWriteDoubleVector1D(&HS_Fz);
g_serializer.readOrWriteDoubleVector1D(&HS_Mrx);
g_serializer.readOrWriteDoubleVector1D(&HS_Mry);
```

```
g_serializer.readOrWriteDoubleVector1D(&HS_Mrz);
//RADIATION FORCES
g_serializer.readOrWriteDoubleVector1D(&rad_Fx);
g_serializer.readOrWriteDoubleVector1D(&rad_Fy);
g_serializer.readOrWriteDoubleVector1D(&rad_Fz);
g_serializer.readOrWriteDoubleVector1D(&rad_Mrx);
g_serializer.readOrWriteDoubleVector1D(&rad_Mry);
g_serializer.readOrWriteDoubleVector1D(&rad_Mrz);
//WAVE FORCES
g_serializer.readOrWriteDoubleVector1D(&wave_Fx);
g_serializer.readOrWriteDoubleVector1D(&wave_Fy);g_serializer.
    readOrWriteDoubleVector1D(&wave_Fz);
g_serializer.readOrWriteDoubleVector1D(&wave_Mrx);
g_serializer.readOrWriteDoubleVector1D(&wave_Mry);
g_serializer.readOrWriteDoubleVector1D(&wave_Mrz);
//thrust in the LSA FRAME
g_serializer.readOrWriteDoubleVector1D(&ThrustLSAx);
g_serializer.readOrWriteDoubleVector1D(&ThrustLSAy);
g_serializer.readOrWriteDoubleVector1D(&ThrustLSAz);
//moment in the COG
g_serializer.readOrWriteDoubleVector1D(&mx);
g_serializer.readOrWriteDoubleVector1D(&my);
g_serializer.readOrWriteDoubleVector1D(&mz);
...
```

C.5 Graphical User Interface

To create the user interface, the source file `QLLTCreatorDialog.cpp` is modified.

Floater Settings tab

In the member function `QLLTCreatorDialog`, a new control panel called *Floater Settings* is created:

```
/*the poliTO floater tab*/

QWidget *floaty_widget = new QWidget ();
tabWidget->addTab(floaty_widget, "Floater Settings");
QHBoxLayout *hBox_floaty = new QHBoxLayout ();
floaty_widget->setLayout(hBox_floaty);
QVBoxLayout *vBox_floaty = new QVBoxLayout;
hBox_floaty->addLayout(vBox_floaty);

QGroupBox *groupBox_floaty = new QGroupBox (tr("Settings "));
vBox_floaty->addWidget(groupBox_floaty);
QGridLayout *grid_floaty = new QGridLayout ();
groupBox_floaty->setLayout(grid_floaty);
int gridRowCount = 0;
```

Floater module ON/OFF

A radio button is added to activate or deactivate the *floater module is added*. It is connected to a variable, `active_floater`, that manages a series of `if` statements.

```
QLabel *label_floaty = new QLabel (tr("Floater: "));
grid_floaty->addWidget (label_floaty, gridRowCount, 0);
QHBoxLayout *miniHBox_floaty = new QHBoxLayout ();
grid_floaty->addLayout(miniHBox_floaty, gridRowCount++, 1);
m_active_floaty = new QButtonGroup(miniHBox_floaty);
QRadioButton *radiobutton_floaty = new QRadioButton ("On");
m_active_floaty->addButton(radiobutton_floaty, 0);
miniHBox_floaty->addWidget(radiobutton_floaty);
```

Change the Wave

The *push button* that opens the window that permits to choose the directory where the wave forces record, the directions array and the array containing the wave record time array is added:

```
label_floaty = new QLabel (tr("Wave Record Directory: "));
grid_floaty->addWidget (label_floaty, gridRowCount, 0);
miniHBox_floaty = new QHBoxLayout ();
grid_floaty->addLayout(miniHBox_floaty, gridRowCount++, 1);
QPushButton *pushbutton_wave = new QPushButton ("Browse");
miniHBox_floaty->addWidget(pushbutton_wave);
miniHBox_floaty->addStretch(0);
connect(pushbutton_wave, SIGNAL(clicked(bool)), this, SLOT(
    loadwavedir()));
```

when the push button is clicked, a `SIGNAL` is emitted. This signal arrives into a slot, that is the member function `loadwavedir`:

```
//polito floater
void QLLTCreatorDialog::loadwavedir(){

    QMessageBox start;
    start.setText("Select a directory containing the waveforce
    records");
    start.exec();

    wavedir_creator = QFileDialog::getExistingDirectory(nullptr, "
    Directory containing Waveforce Records", "/home" ,QFileDialog::
    ShowDirsOnly | QFileDialog::DontResolveSymlinks);
}
```

Initial Conditions

Some *Number Edit* boxes are added to set the coordinates of the COG in the FRA frame, the rotations of the floater around the FRA axes and the COG-turbine base distance:

```
label_floaty = new QLabel (tr("COG x position [m] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_x = new NumberEdit ();
floaty_x->setMaximumWidth(EditWidth);
floaty_x->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_x);
grid_floaty->addWidget(floaty_x);

label_floaty = new QLabel (tr("COG y position [m] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_y = new NumberEdit ();
floaty_y->setMaximumWidth(EditWidth);
floaty_y->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_y);
grid_floaty->addWidget(floaty_y);

label_floaty = new QLabel (tr("COG z position (wrt SWL) [m] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_z = new NumberEdit ();
floaty_z->setMaximumWidth(EditWidth);
floaty_z->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
+miniHBox_floaty->addWidget(floaty_z);
grid_floaty->addWidget(floaty_z);

label_floaty = new QLabel (tr("COG rx position [deg] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_roll = new NumberEdit ();
floaty_roll->setMaximumWidth(EditWidth);
floaty_roll->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_roll);
grid_floaty->addWidget(floaty_roll);

label_floaty = new QLabel (tr("COG ry position [deg] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_pitch = new NumberEdit ();
floaty_pitch->setMaximumWidth(EditWidth);
floaty_pitch->setMinimumWidth(EditWidth);
```

```
miniHBox_floaty = new QHBoxLayout ();
 miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_pitch);
grid_floaty->addWidget(floaty_pitch);

label_floaty = new QLabel (tr("COG rz position [deg]"));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_yaw = new NumberEdit ();
floaty_yaw->setMaximumWidth(EditWidth);
floaty_yaw->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_yaw);
grid_floaty->addWidget(floaty_yaw);

label_floaty = new QLabel (tr("COG-Turbine Base Distance [m] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_COG = new NumberEdit ();
floaty_COG->setMaximumWidth(EditWidth);
floaty_COG->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_COG);
grid_floaty->addWidget(floaty_COG);
```

"Wave time steps" and number of directions

The *Number Edit* boxes that permits to set the number of time steps by which the wave force record is composed and the number of directions considered are added:

```
label_floaty = new QLabel (tr("Number of timesteps "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_nt = new NumberEdit ();
floaty_nt->setMaximumWidth(EditWidth);
floaty_nt->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_nt);
grid_floaty->addWidget(floaty_nt);

label_floaty = new QLabel (tr("Number of directions "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_ndir = new NumberEdit ();
floaty_ndir->setMaximumWidth(EditWidth);
floaty_ndir->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_ndir);
```

```
grid_floaty->addWidget(floaty_ndir);
```

Incoming wave direction

A *Number Edit* box that permits to choose the direction of the incoming wave is added:

```
label_floaty = new QLabel (tr("Incoming Wave Direction [deg] "));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floaty_incoming = new NumberEdit ();
floaty_incoming->setMaximumWidth(EditWidth);
floaty_incoming->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floaty_incoming);
grid_floaty->addWidget(floaty_incoming);
```

Mooring Lines Controls

Some *Number Edit* boxes that permit to edit the mooring lines-related parameters are added:

```
label_floaty = new QLabel (tr("Moorings Stiffness [N/m]"));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floatymoor_K = new NumberEdit ();
floatymoor_K->setMaximumWidth(EditWidth);
floatymoor_K->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floatymoor_K);
grid_floaty->addWidget(floatymoor_K);


label_floaty = new QLabel (tr("Moorings Weight Force [N]"));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floatymoor_W = new NumberEdit ();
floatymoor_W->setMaximumWidth(EditWidth);
floatymoor_W->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floatymoor_W);
grid_floaty->addWidget(floatymoor_W);

label_floaty = new QLabel (tr("Moorings length [m]"));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floatymoor_L = new NumberEdit ();
floatymoor_L->setMaximumWidth(EditWidth);
```

```
floatymoor_L->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floatymoor_L);
grid_floaty->addWidget(floatymoor_L);

label_floaty = new QLabel (tr("Moorings pre-tensioning [N]"));
grid_floaty->addWidget(label_floaty, gridRowCount++, 0);

floatymoor_preT = new NumberEdit ();
floatymoor_preT->setMaximumWidth(EditWidth);
floatymoor_preT->setMinimumWidth(EditWidth);
miniHBox_floaty = new QHBoxLayout ();
miniHBox_floaty->addStretch();
miniHBox_floaty->addWidget(floatymoor_preT);
grid_floaty->addWidget(floatymoor_preT);
```

Appendix D

Aerofoils Aerodynamic Tables

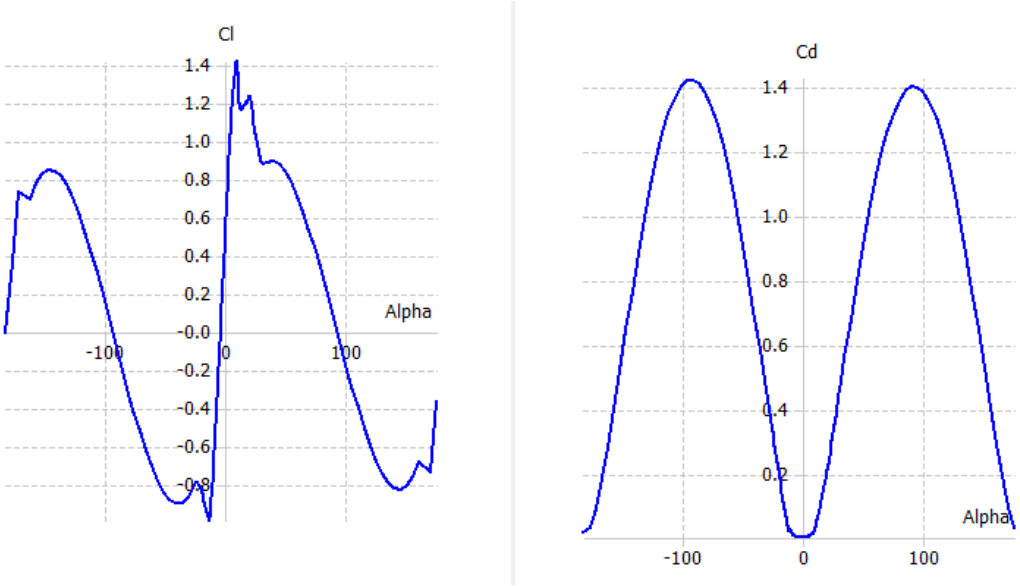


Figure D.1: *DU91W2250LM Aerodynamic Tables*

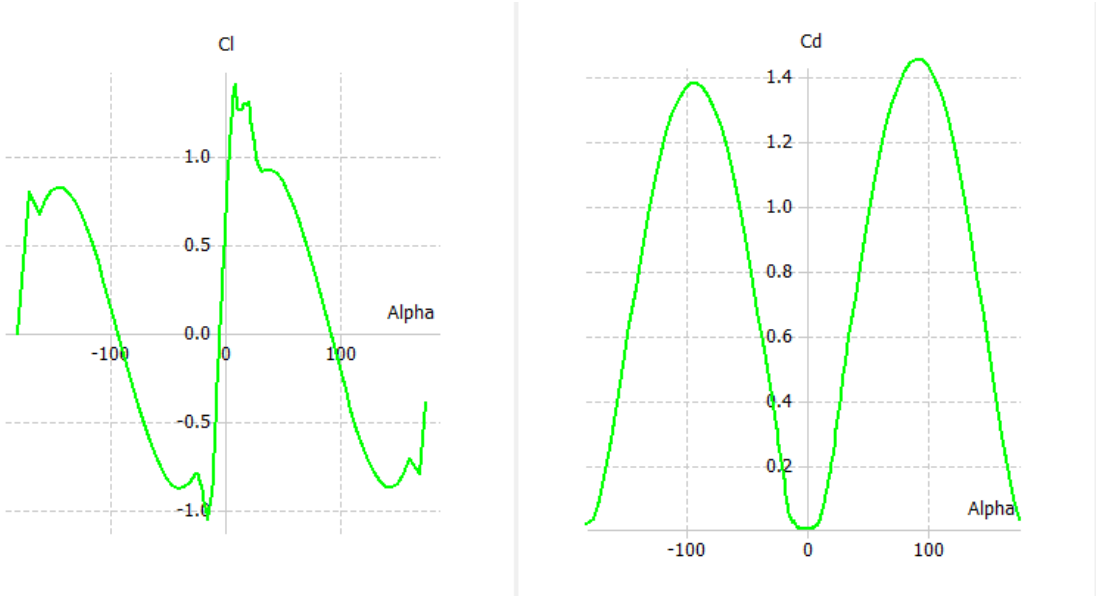


Figure D.2: *DU93W210LM Aerodynamic Tables*

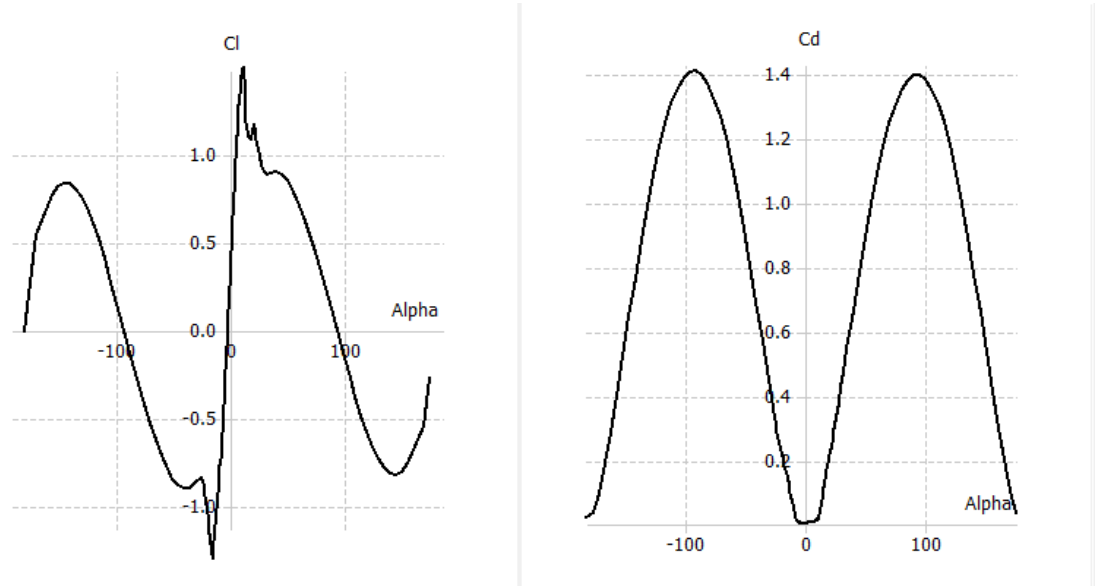


Figure 6.3: *DU97W300LM Aerodynamic Tables*

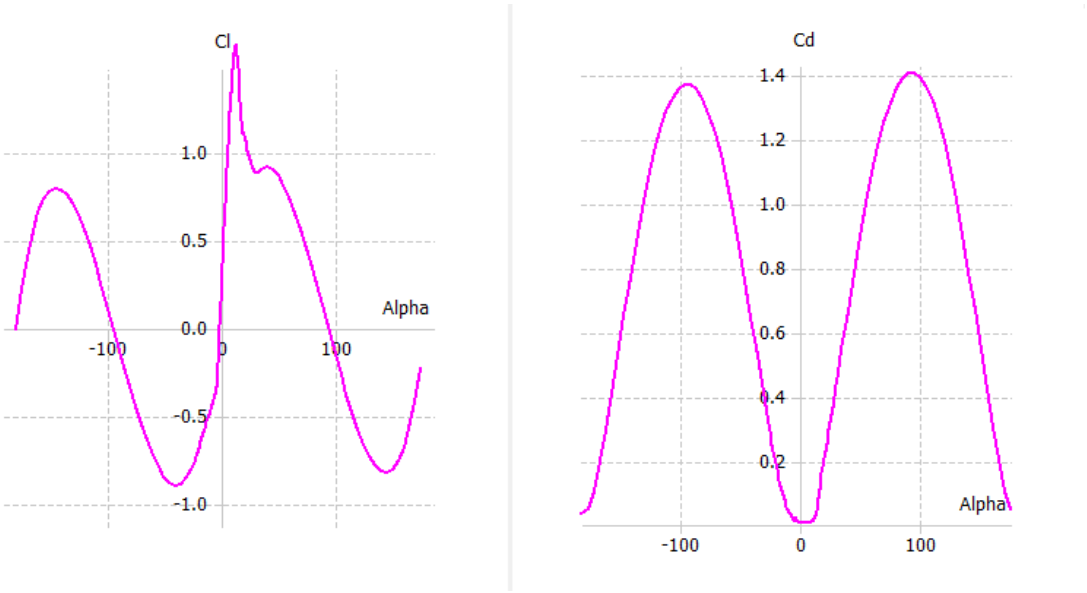


Figure 6.4: *DU99W350LM Aerodynamic Tables*

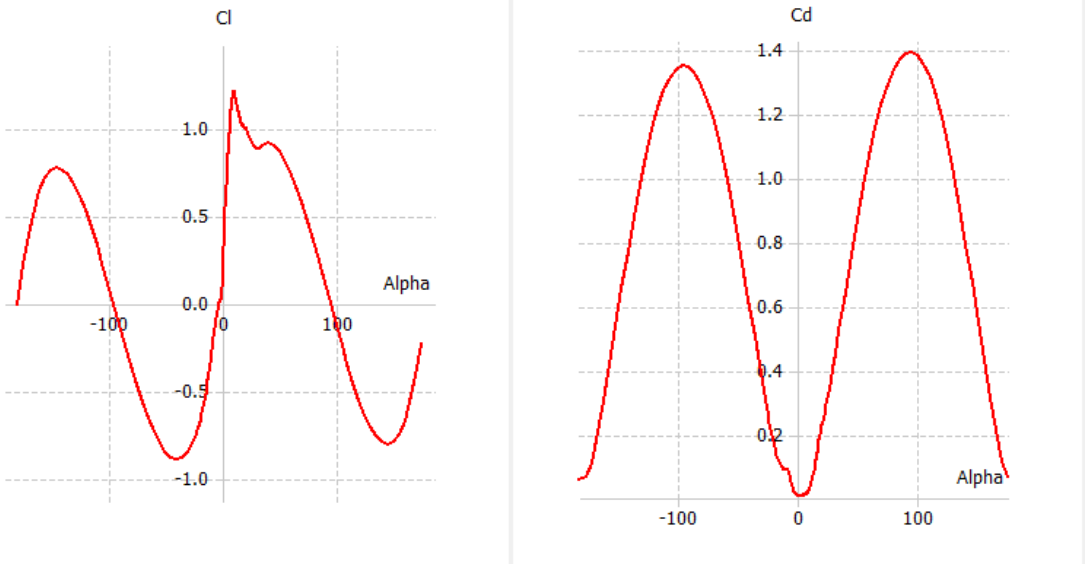


Figure 6.5: *DU99W405LM Aerodynamic Tables*

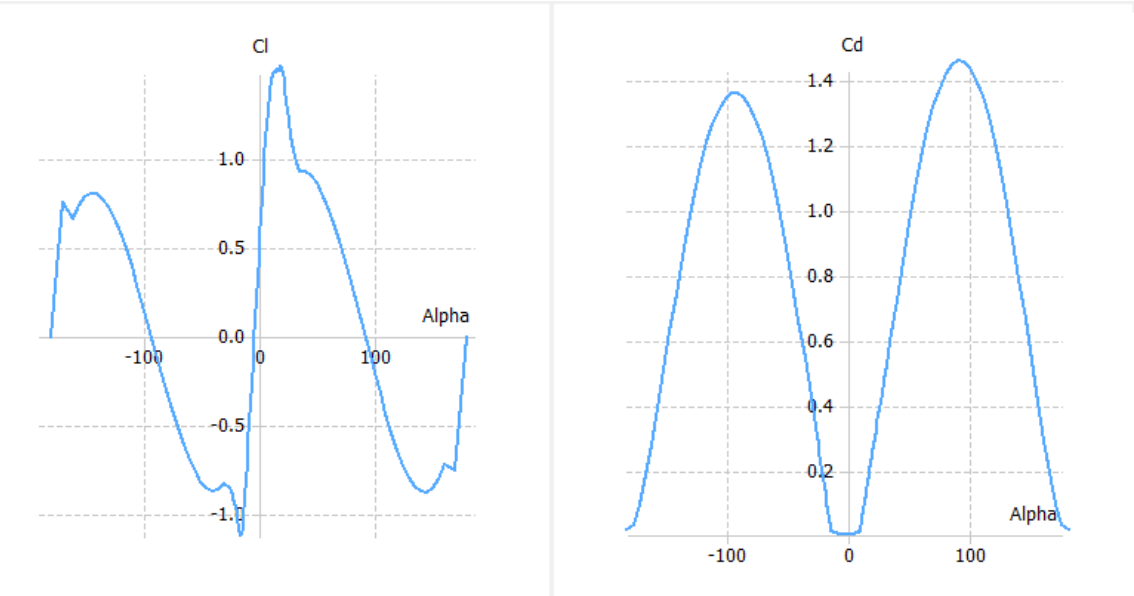


Figure 6.6: *NACA64618 Aerodynamic Tables*

Bibliography

[1] USGCRP, 2017, *"Climate Science Special Report: Fourth National Climate Assessment"*, Volume I [Wuebbles, D.J., D.W. Fahey, K.A. Hibbard, D.J. Dokken, B.C. Stewart, and T.K. Maycock (eds.)]. U.S. Global Change Research Program, Washington, DC, USA, 470 pp, doi:10.7930/J0J964J6.

[2] IRENA (2016), *"Innovation Outlook: Offshore Wind"*,International Renewable Energy Agency, Abu Dhabi

[3] Morten Thøtt Andersen, *"Floating Foundations for Offshore Wind Turbine"*, Ph.D. Dissertation, Aalborg University Press, 2016.

[4] Carbon Trust, *"Floating Offshore Wind: Market and Technology Review"*, Technical report, Prepared for the Scottish Government, 2015.

[5] *"Lazard's levelized cost of energy analysis"*, Lazard Ltd., version 12.0, 2018. [Online]. Available: `https://www.lazard.com/media/450784/lazards-levelized-cost-of-energy-version-120-vfinal.pdf`.

[6] ITIC, presented at the 17th International Towing Tank Conference, Goteborg, 1984.

[7] J. Journée and W. Massie, *"Offshore Hydromechanics"*, Delft University of Technology, 2001.

[8] W. Cummins, *"The impulse response function and ship motions"*, DTIC Document, Tech. Rep., 1962.

[9] Kapil Ghosh, Md. Quamrul Islam, Mohammad Ali, *"Pressure Distributions and Forces on Hexagonal Cylinder"*, Procedia Engineering, Volume 105, 2015, Pages 835-843 ,ISSN 1877-7058, doi:10.1016/j.proeng.2015.05.096, `http://www.sciencedirect.com/science/article/pii/S1877705815008930`

[10] A. Robertson, J. Jonkman, M. Masciola, H. Song, A. Goupee, A. Coulling and C. Luan, *"Definition of the semi-submersible floating system for phase II of OC4"*, *Offshore Code Comparison Collaboration Continuation (OC4) for IEA Task*, vol. 30, 2012. Available: `http://www.nrel.gov/docs/fy14osti/60601.pdf`.

[11] ANSYS, Inc., *Aqwa Theory Manual*, Release 15.0, 2013.

[12] T. Ogilvie, *"Recent progress towards the understanding and prediction of ship motions"*, *Proceedings of the 5th Symposium on Naval Hydrodynamics*, 3-79, 1964.

[13] Mutidieri, Luigi, *"Dynamic modelling and analysis of an offshore floating wind turbine"*, supervised by Giuliana Mattiazzo, Giovanni Bracco, Nicola Pozzi, Giacomo Vissio. Politecnico di Torino, March 2017.

[14] T. Pérez and T. I. Fossen, *"Time-vs. frequency-domain identification of parametric radiation force models for marine structures at zero speed"*, *Modeling, Identification and Control*, vol. 29, no. 1, pp. 1-19, 2008.

[15] T. Pérez, *"Identification of dynamic models of marine structures from frequency-domain data enforcing model structure and parameter constraints"*, ARC Centre of Excellence for Complex Dynamic Systems and Control, pp. 1-28, 2009.

[16] T. Pérez and T. I. Fossen, *"A matlab toolbox for parametric identification of radiation-force models of ships and offshore structures"*, Modeling, Identification and Control, Vol. 30, No. 1, pp.1-115, ISSN1890-1328

[17] D. Marten, *"QBlade v0.95. Guidelines for Lifting Line Free Vortex Wake Simulations"*, 2016

[18] P.G. Saffman, *"Vortex Dynamics"*, Cambridge University Press, 1992

[19] A. van Garrel, *"Development of a wind turbine aerodynamics simulation module"*, Technical Report ECN-C-03-079 EN, Energy research Centre of the Netherlands, 2003.

[20] Himmelskamp H. *"Profile investigation on a rotating airscrew"*. PhD thesis, Gottingen University: PhD Dissertation, 1945.

[21] G.Bangga, T.Lutz, E. Kramer, *"An Examination of Rotational Effects on Large Wind Turbine Blades"*, 11th EAWE PhD Seminar on Wind Energy in Europe, September 2015.

[22] Fincantieri Offshore, Sea Flower, Fincantieri S.p.A., 2016. [Online]. Available: `https://www.fincantierioffshore.it/sea-flower.html` (visited on 29/10/2018).

[23] J. Jonkman, S. Butterfield, W. Musial, and G. Scott, *"Definition of a 5-MW reference wind turbine for offshore system development"*, National Renewable Energy Laboratory, Golden, CO, Technical Report No. NREL/TP-500-38060, 2009.

[24] O. DeAndrade and A. Duggal, *"Analysis, Design and Installation of Polyester Rope Mooring Systems in Deep Water"*, OTC, vol. 20833, pp. 3-6, 2010.

[25] S. Barth, P.J. Eecen, *Description of the relation of Wind, Wave and Current Characteristics at the Offshore Wind Farm Egmond aan Zee (OWEZ) Location in 2006*, ECN-E-07-104.