Executive Summary

Wind turbines with floating support structures are blessed with their potential in generating high quality and affordable electricity due to the economics of scale. However, conventional land-based wind turbines blade pitch control system coupled with a floating structure present a problem known as control-induced resonance. The problem, if unresolved may lead to devastating implications such as structural and machinery failure due to fatigue.

This thesis started off by examining the working theory of a detuned version of conventional land-based controller intended for offshore floating wind turbines (FOWT) and subsequently using it as a baseline for performance comparison with a few alternative state of the art designs that also aim at reducing control-induced resonance. Rather than developing novel controller designs, the focus of this work has been on code implementation and extending the performance evaluation criteria beyond structural. A decoupled approach is taken to separately account for the dynamics of different component levels. Modifications to the baseline control strategy are made through a java interface interacting with aero-hydro-servo-elastic code SIMA which is the environment where the global analysis is carried out. Finally the global response is then imported to MBS simulation software SIMPACK for drivetrain multi-body dynamics simulations.

Three alternative controllers are considered and implemented in this work. The first one is simple yet elegant variable speed variable pitch strategy proposed by Lackner in which platform surge velocity is being fed to the control loop with a constant gain as active damping term to augment the reference speed of the rotor [1]. The 'energy shaping' filter developed by Pedersen in his doctoral thesis augment the reference speed through a function developed on the basis of energy conservation [2]. Lastly the energy shaping controller is complement with an individual pitch control mechanism to reduce blade flapwise bending load described by Lackner [3] and Bossanyi [4].

The performance of controllers is evaluated based not only on their ability to mitigate structural loads but also loads on drivetrain component in which limited studies has been made. Frequency response analysis proves to be effective in identifying frequencies at which the controllers are behaving poorly and allows the possibility of filtering out of such frequencies at input. By looking at the frequency response of the component in the drivetrain, sources of excitation from global analysis can also be identified and thus giving a more complete picture with regards to the gains and losses using a particular controller. Finally, one hour fatigue damage comparison is used to justify that the mitigation of loads at certain frequencies leads to an improved fatigue performance.

Preface

This thesis is performed at Department of Marine Technology at Norwegian University of Technology and Science (NTNU) and is carried out during the final semester of a 2-year Nordic Master in Maritime Engineering program in fulfillment of the mandatory course TMR4930 Marine Technology, Master's Thesis.

The basis for this topic stemmed initially from my genuine interest in marine control and the desire to study the effects of different control strategies on stability and structural performance of floating marine structures. As the marine industry transits into the age of digital with increased amount of sensors used on-board, the change presents itself as an opportunity for improved vessel performance and cost reduction. Floating offshore wind turbines belong to a category that is particularly well positioned to tap into the advantage that comes with the industrial shift. With research efforts pouring into the wind industry in pace that we have never seen from developing better materials to new installation methods, I feel the need to emphasize on the capability and the beauty of control system in achieving structural load reduction with a relatively smaller change to existing hardware. It is therefore my passion to implement various load mitigating control strategies and evaluate their performance not only in the conventional structural point of view but also the implications on drivetrain performance, which I think have not been given enough attention to.

In truth, I could not have achieved the current level of success in my work without the support from many and I owe a great debt to those who have helped me along the journey. To Erin Bachynski, the supervisor of my thesis work, I would like to thank you for the insightful discussions, guidance and most importantly for influencing me with your passion in wind turbine. It has been a rewarding experience to study under your supervision for the past one year. Amir Rasekhi Nejad, the co-supervisor of my thesis work, thank you for the knowledge in drivetrain technology and always being so patient in guiding the second part of my work.

To Morten Dinhoff Pedersen, Associate Professor at the Department of Engineering Cybernetics, thank you for equipping me with the fundamental control knowledge required in the thesis and going out of your way to explain the working theory of 'energy shaping' controller in such details. I wish to also extend my thanks to PhD candidate Carlos Eduardo Silva de Souza for the immense help with the familiarization of SIMA. My classmate and project partner Xuwen Wang has also been very supportive. Thanks for the project work, study sessions and our exchange of questions to one another on wind turbine. I have learned a great deal from such experience.

Finally to the people that I am ever indebted to, my family, the past two years would not have been so smooth without your support. I look forward to next phase of my life and you will always be an integral part of it.

Table of Contents

List of Tables

List of Figures

Abbreviations

a	axial induction factor [-]
a'	radial induction factor [-]
C_D	aerofoil drag coefficient [-]
C_L	aerofoil lift coefficient [-]
C_M	aerofoil moment coefficient [-]
C_t	aerofoil tangential force coefficient [-]
C_n	aerofoil normal force coefficient [-]
α	angle of attack [°]
σ	solidity of rotor [-]
ψ	stream function $[m^2/s]$
U_∞	free stream velocity $[m/s]$
q	excitation force $[kN/m]$
v	relative velocity between body and wind $[m/s]$
D	characteristic length $[m]$
F	drag force $[kN]$
m	body mass $[kg]$
K	hydrostatic stiffness $[kN/m]$
C	potential damping $[kN/(m/s)]$
B_1	linear damping $[kN/(m/s)]$
B_2	quadratic damping $[kN/(m/s)^2]$
ω	wave frequency $[rad/s]$
ζ	wave elevation $[m]$
ζ_a	wave amplitude $[m]$
k	wave number $[1/m]$
ϕ	wave velocity potential $[m^2/s]$
F_rad	excitation force due to radiation $[kN]$
F_diff	excitation force due to diffraction $[kN]$
H_S	significant wave heigh $[m]$
T_P	wave peak period $[s]$
U	wind speed $[m/s]$
I	turbulence intensity [-]
$F_{x,y,z}$	force applied on drivetrain main shaft in x, y and z directions in drivetrain coordinate $[kN]$
$M_{x,y,z}$	moment applied on drivetrain main shaft in x, y and z directions in drivetrain coordinate $[kNm]$
$F_{TowerTop_{x,y,z}}$	force on tower top in x, y and z directions in tower local coordinate $[kN]$
$M_{TowerTop_{x,y,z}}$	moment on tower top in x, y and z directions in tower local coordinate $[kNm]$
θ	blade pitch angle [°]
Ω	rotor speed $[rad/s]$
Ω_{ref}	reference rotor speed $[rad/s]$
Ω_0	nominal rotor speed $[rad/s]$
ω_n	natural frequency of system $[rad/s]$
J	rotor moment of inertia $[kgm^2]$
b_d	coefficient of power dissipation

Chapter 1

Introduction

1.1 Background

Wind power has developed into a reliable and competitive source of energy driven by rapid technological revolutions and increased environmental awareness. It is renewable, non-pollutant and therefore according to World Wind Energy Association [13] - the overall capacity of wind turbines installed worldwide reached 600 GW by the end of 2018 with 53,900 MW added in 2018 alone. This accounts to nearly 6% of the global electrical energy demand. In Europe, wind turbine manufacturing industry supports over 260,00 highly-skilled jobs and generates a turnover of 60 billion Euros per annum. Being one of the leaders in wind technology development, the European wind industry exports 8 billion Euros per annum in technology and services [14].

The first offshore wind farm was commissioned in Denmark in 1991 and since then the number has grown drastically. It is predicted that by 2022, global operating capacity of offshore wind farms will reach 46.4 GW of which 33.9 GW of the capacity being operated in Europe, 11.3 GW in Asia and 1.2 GW in North America [15]. Concrete foundations are used in the more benign Baltic Sea sites; monopiles have been used in shallow seas while non-monopile steel structures (jackets, tripods and tripiles) are preferred in deeper waters up to 40 meters water depth. Despite innovative designs, bottom founded structures present limitations due to the scarcity of shallow coastal waters with high quality wind and public preference to keep wind turbines far offshore. Japan and the United States are in particular exposed to such limitations due to the lack of shallow sites. Floating solutions come in as an alternative offering the offshore wind industry access to wind resources in water depths greater than 50 meters. In transitional water depths (30-50 meters), floating foundation may offer lower cost solution than fixed bottom foundations due to the potential standardization of foundation designs, capability of onshore assembly and the used of available installation vessels in the market. In the long term, the industry recognizes the potential of floating in water depths where bottom founded structures are no longer cost efficient. Following the successful deployment of a full scale prototype Hywind demonstration unit in 2009, Equinor successfully launched the world's first commercial wind farm with five 6 MW turbines offshore Scotland in 2017 marking a key milestone to the

development of floating wind solutions.

The key to reliable floating offshore wind turbines lies in the design of the floating support structures according to site specific environmental conditions. According to IEC 61400-3 design requirement, the load on a offshore wind turbine has to be calculated through integrated dynamic analysis taking into consideration characteristic wind, hydrodynamic and permanent loads [16]. Time domain aero-hydro-servo-elastic analysis is therefore typically performed on floating offshore wind turbine (FOWT) to take into account the effect of nonlinearities in aerodynamic, structural dynamic and control system. Some examples of computer aided engineering (CAE) tools being used are such as FAST by NREL, Bladed-Sesam by DNVGL and SIMA by SINTEF Ocean which are all based on coupled aero-hydro-servo-elastic simulation in time domain. Simulations using these CAE tools can be performed within a relatively short amount of time and therefore is suited for preliminary design where many load cases need to be tested within a short time frame.

More sophisticated Multibody Dynamics (MBD) software such as Adams and SIMPACK were used to capture complex interactions between disciplines including motion, structures, actuation, and controls. These MBD simulation software aim to produce as close as possible the real world behavior by including more degree of freedoms (DOF) and using a higher time step resolution as compared for example, the FEM solver in SIMA. Such high fidelity simulations require significantly longer computing time and therefore it is not possible in this project to employ these software on every load case. However, in order to study in detail the response on the component-level of the generator, global response of selected load cases generated from coupled analysis were imported to an MBD software for further analysis. The global analyses for a range of load cases of the platform in this project were performed using SIMA while the detailed analyses of the generator for selected load cases were perform in SIMPACK environment.

Traditionally, blade pitch controllers are used to regulate the wind turbine aerodynamic power above rated speed. As wind turbines become larger, there is an increase of interest in the load mitigating ability of these controllers. This work explores the implications of different load mitigating control strategies employed on a spar-supported FOWT through modifying an existing controller and performing load analyses. The results are then compared to the baseline for performance evaluation of the developed controller.

1.2 Previous Work and State of the Art Technology

Numerous studies have been carried out to analyze the performance of different control strategies with Bossanyi [17] being one of the earliest work to describe various conventional and recently developed control algorithms. Without referencing to specific type of supporting structure, Bossanyi's work described the conventional Proportional-Integral (PI) controller and showed that it can be modified to include more terms and achieves reduction in motion responses. Methods such as the use of additional sensors as to provide more inputs to the PI-controller and the use of more advanced observer-state estimator control algorithm to optimize controller performance are also briefly described. Control-induced negative damping of tower motion above rated wind speed was first described by Nielsen et. al. [18] where conventional constant power blade pitch control strategy. Sim-

ulations and model scale experiments on on Hywind floating concept show that at above rated wind speed clear resonance at surge and pitch frequency were detected. It was also mentioned that the addition an "active damping" component successfully reduced the peak of resonance. However no further details were given on the design of the controller. Several subsequent work addressed the negative damping problem and attempted to improve existing control strategies. Larson and Hanson [19] for example tested a few controller natural frequency ranging from 0.01 to $0.1Hz$ and discovered that the controller worked best to reduce large motion excitation at lower frequencies. A slower to react controller however resulted in higher speed variation of the rotor and undesirably lower power quality. In another research, Jonkman [20] studied the effect on the pitch motion damping of a barge supported wind turbine using three different methods. The first method was to include tower-top acceleration as an input in an additional blade pitch control loop as suggested by Bossanyi [17]. The second method was to use variable blade pitch to stall instead of feather as in conventional controllers and the third method was to detune the gains in the variable blade pitch to feather controller. Detuning gains showed the most promising of results but did not, however reduce the platform pitch negative damping problem sufficiently enough. Lackner [1] presented a simple controller that uses platform pitch velocity as an input to augment the generator speed set point. This easily implemented controller showed significant reduction of motion with a slight downside of increased generator power and speed variability. Lackner further commented that this increase in power variability might not amount to much effect as compared to the overall power variability of an entire wind farm. Similarly, in their patented design Skaare and Nielsen [8] injected an active damping term by passing the platform pitch velocity through a "black box" (that is said to filter and process the signal) to augment the generator reference speed. A block diagram of the design is shown in Figure 1.1.

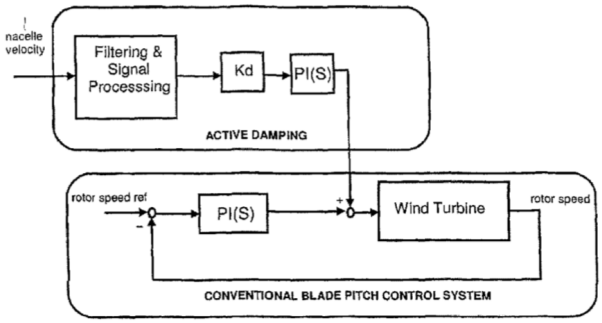

Figure 1.1: Hywind patented controller block diagram [8]

In his Phd. thesis, Pedersen [2] proposed an energy shaping controller motivated by the conservation of energy within the wind turbine system. Pedersen showed that the energy equilibrium of a floating offshore wind turbine can be represented using a dynamic equation with perturbation in the form of change in wind speed, δw and change in rotor speed, $\delta\Omega$. This idea allowed the derivation of a variable speed controller mathematical

model that uses δw as input to depict a corresponding desired $\delta\Omega$. This method is pivotal to this work and will be examined closely in Chapter 5. Finally in a more recent work, Fleming et. al. [21] simultaneously used platform pitch angle and nacelle velocity as inputs to augment the generator reference speed.

Other than collective pitch controllers (CPC), work has also been done exploring the advantages of controlling blade pitch individually. Bossanyi [4] detailed the application of individual pitch controller (IPC) using linear quadratic gaussian (LQG) for multivariable optimization and a simple proportional integral (PI) controller when the control problem is simplified to a single input single output (SISO) problem. Lackner [3] further documented in details the implemention of a SISO linear time invariant (LTI) IPC on a 5 MW turbine using aeroelastic codes. Significant reduction of blade root flapwise damage equivalent load (DEL) was achieved for wind speed above rated with little change on the power production but a much higher blade pitch activity.

Further into the performance of drivetrain, Nejad et. al. [22] investigated the fatigue damage of mechanical components of a 5 MW land-based turbine drivetrain used on floating wind turbines. Significant increase in damage was observed for main bearings that are carrying axial load (induced by pitch and surge motions) as compared to land-based turbines. The drive train on a spar platform appeared to suffer a much higher main bearing damage as the environmental condition became more severe.

While aeroelastic codes are typically used to study the preliminary performance of controllers, work has been done using integrated aeroelastic-drivetrain dynamics code to better model the behavior of drivetrains. Girsang et. al. [23] for example, used the integrated FAST-Simscape to accurately capture resonant excitations of the drivetrain. Their work proposed introducing virtual inertia in the compensating torque to modify the eigenfrequency to avoid resonance.

1.3 Floating Offshore Wind Turbines

FOWT with single turbine configurations can be divided into 3 main categories namely ballast stabilized FOWT, buoyancy stabilized FOWT and mooring line stabilized FOWT. Spar platforms are typical examples for ballast stabilized FOWTs. A spar platform stabilizes itself via a heavy ballasted cylinder moored by catenary or taut lines. A mooring stabilized FOWT such as a tension-leg platform (TLP) achieves stability through maintaining tension in the taut mooring lines between the hull and the anchors. A barge platform is generally kept in position using catenary mooring and stays afloat mainly through the hull's buoyancy which make it a type of bouyancy stabilized FOWTs. A schematic of the 3 common types of platforms are shown in Figure (1.2).

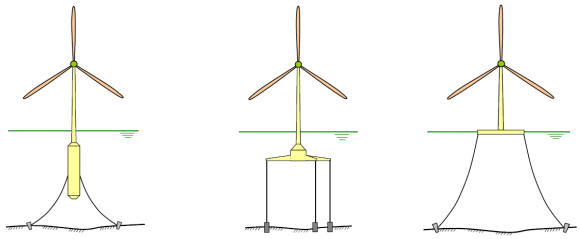

Figure 1.2: Typical floating wind solutions (from left): Spar platform, TLP and barge platform

The dynamic coupling of wind turbines with floating structures has always poses challenge to the successful development of commercialized floating wind farm. At above rated conditions, a floating platform experiences excessive load due to large pitch motions. While each type of platform has it own advantages and disadvantages, choosing a suitable platform based on the operating sites and the environmental conditions is important to ensure a good return of investment. In Table 1.3, Van and Biswajit compared the advantages and disadvantages of the 3 types of floating wind turbines and it shows that for deep water, spar type platforms have superiority over TLP due to the cheaper mooring cost and simpler hull design [9].

Spar (S-FOWT)	TLP- FOWT	Barge (B-FOWT)
• Small water line area: Buoyancy far below free surface, stability relies on buoyancy/weight distribution • Little volume close to free surface –small wave forces • Catenary mooring: Low cost and easy to install. • Inexpensive platform geometry • Installation need special procedures • Advantageous in natural period, anchors, operation and maintenance; challenging in weight and mooring • Not suited for shallow water.	• Small water line area: Buoyancy far below free surface, stability relies on positive mooring tension • Little volume close to free surface: Small wave forces • Vertical moorings: Positive tension needed, expensive anchors, weight sensitive. • Complexity of platform depends upon design, • Advantageous in natural period, couple motions, operation and maintenance; challenging in anchors and construction • Not suited for shallow water.	• Large water plane area: Buoyancy and stability • Large volume close to free surface: Large wave forces • Conventional mooring lines: Easy to install; large motions implies large forces. • Simpler, inexpensive platform • Deck convenient for operation. • Advantageous in weight, anchors; challenging in natural period, operation and maintenance • Easy to install, shallow water capability, suitable to calm sea (e.g. the Mediterranean)

Figure 1.3: Comparison of the advantages and disadvantages for 3 common types of floating platforms [9]

1.4 Project Objectives

The main objectives of this study is to provide a better understanding of the impact of various load mitigating control strategies through decoupled analyses. As there is a lack

of work on transient behaviors of controllers, it is intended that simple load cases such as step wind velocity being used to give more insight into these transient behaviors. Many recent work on controller performance focus mainly on comparing tower and blade fatigue damage through global analysis. Therefore, this work carries the responses from global analysis to multi-body dynamics simulation environment to explore the effect on the component level of a wind turbine drivetrain. Finally, it is expected that this study is able to reproduce results that agree with previous work in terms of load mitigation performance and to highlight any trade-off that arises in exchange.

The NREL 5 MW wind turbine control strategy as described by Jonkman et. al. is first setup as the baseline for performance comparison. Next extensive environment load case (ELC) simulations is performed in SIMA to obtain the global response of the model. The ELCs involved are based on the work by Nejad et. al. [22] with only the ELCs above rated wind speed investigated. The control strategies featured in the simulations are the NREL 5 MWs baseline controller, baseline controller with active damping (using nacelle surge velocity), energy shaping controller developed by Pedersen [2] and finally the energy shaping controller will be used simultaneously with the IPC.

Structural response improvement such as tower bending moments, blade root bending moments in the form of 1-hour damage equivalent load (DEL) are described and compared. The resulted wind turbine characteristic ie. the power quality, speed excursion and blade actuator activities are presented and discussed. The response from global analysis is then used to imported to SIMPACK for detailed analysis of the generator drivetrain. Bearing and gear 1-hour damage are compared for different controllers.

Chapter 2

Theory

Floating offshore wind turbines are designed to be operated in water depths that exceed 60 meters and are usually exposed to harsher environmental conditions as compared to fixed offshore wind turbines due mainly to the larger volume and motions of the floating structures. According to IEC 61400-3-2 [24], the loads are categorized into gravitational and inertia loads, aerodynamic loads, hydrodynamic loads, sea/lake ice loads and other relevant loads such as mooring loads and wake effects from neighbouring wind turbines. Figure 2.1 illustrates typical loads that are acting on a FOWT. In modelling terms, it is convenient to divide an aero-hydro-servo-elastic model into two main parts - structural model and external load model. The structural model describes in time and space the initial boundary conditions of all structural elements and the physics that governs the load displacement relationship. The external load model describes both aerodynamics and hydrodynamic loads that are exerted on the structural model. The remaining sections of this chapter will give an overview of the various loads that are relevant in the design of a FOWT. Theory of load calculations will be covered briefly with the intention to focus on the simplification to actual physics that SIMA's aero-hydro-servo-elastic code uses and the limitations resulted from such simplifications.

2.1 Aerodynamic Model

The aerodynamic loads applied to the blades can be separated into a steady component and one which is due to wind speed fluctuation (i.e. turbulence). The load component that varies periodically due to a steady spatial variation of wind speed over the rotor swept area is deterministic in nature. It is a function dependent on parameters such as hub height, wind speed, rotor speed and wind shear. The random load component that is due to wind speed fluctuations involves uncertainties, which has to be characterized using probability distributions. This load component is termed stochastic load component.

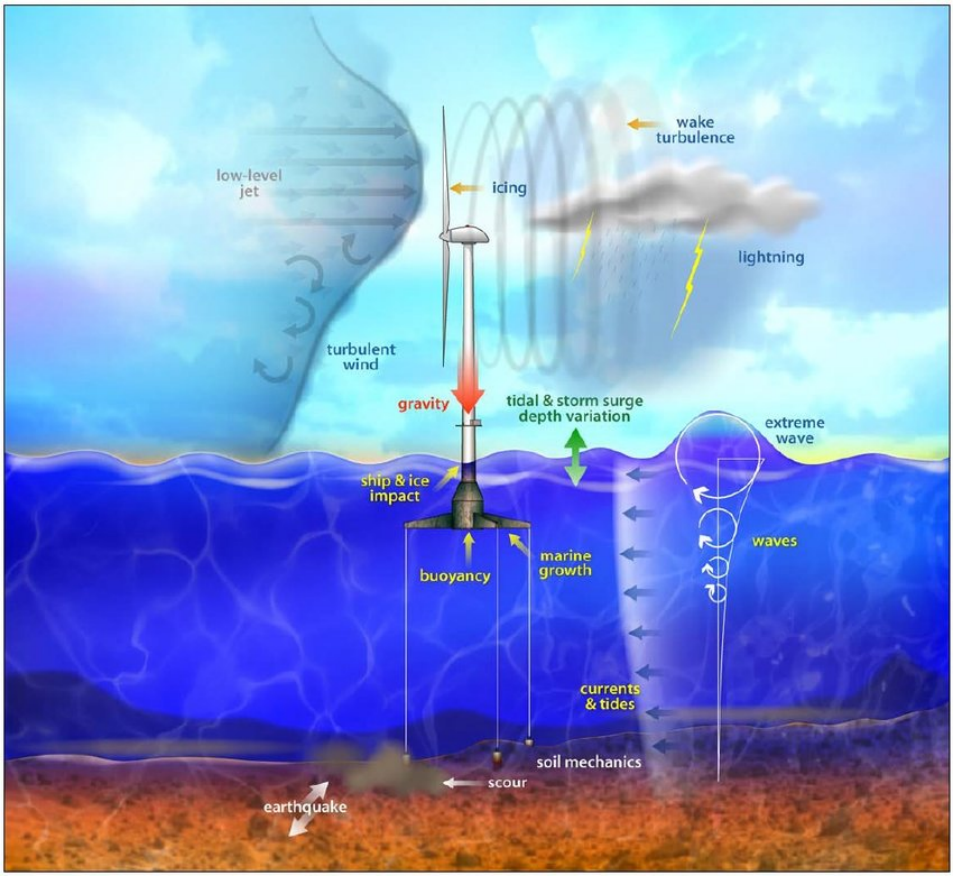

Figure 2.1: Load on an offshore wind turbine [6]

2.1.1 Deterministic Aerodynamic Loads on Blade

The deterministic aerodynamic loads are calculated using the blade element momentum (BEM) method which is a combination of aerofoil dynamics and 1-D momentum theory on an arbitrary control volume around a wind turbine as shown in Figure 2.2.

A typical execution of BEM code can be generalized as follows,

1. initialize starting guess of a and a'
2. calculate ϕ and α
3. look up C_D and C_L using the computed α
4. update a and a' and check if the convergence criteria is satisfied

where

- C_L is the aerofoil lift coefficients

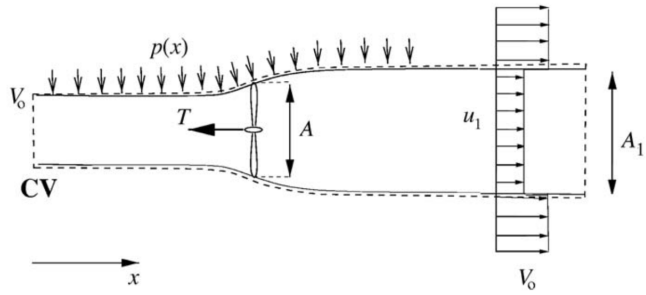

Figure 2.2: Control volume of wind turbine flow momentum [10]

- C_D is the aerofoil drag coefficients
- ϕ is the angle between relative wind velocity and the rotor plane
- α is the angle of attack

a and a' are the axial and radial induction factors respectively and they can be calculated through

$$a = \frac{1}{4\frac{F\sin^2\phi}{\sigma C_n} + 1} \tag{2.1}$$

$$a' = \frac{1}{4\frac{F\sin\phi\cos\phi}{\sigma C_t} - 1} \tag{2.2}$$

2.1.2 Stochastic Aerodynamic Loads on Blade

Stochastic aerodynamic loads are due to a random variation of wind velocity from its mean value with respective to time and space. Such turbulence introduced as a result, random fluctuation blade loads. To analyze stochastic aerodynamic loads, a stochastic turbulence wind field simulator is required to simulate a wind field covering the rotor disc area. This is achieve through generating random wind velocity time series block around the rotor disc that consists of user-defined statistical properties. The time series is characterized by standard power spectra such as Von Karman and Kaimal.

Tower Shadow

Tower shadow effect occurs at region close to the tower due to the reduction of wind speed. The presence of the tower causes a disruption of the otherwise smooth air flow and the effect is more significant for tubular towers than for lattice towers due to bigger wind area. For a tubular tower, the wind velocity reduction upwind can be estimated using potential flow theory [11]. Through placing a doublet in a uniform flow, the stream function can be expressed as,

$$\psi = U_\infty y \left(1 - \frac{(D/2)^2}{x^2 + y^2}\right) \tag{2.3}$$

where D is the tower diameter and x and y are the coordinates with respect to the center of tower as shown in Figure 2.3.

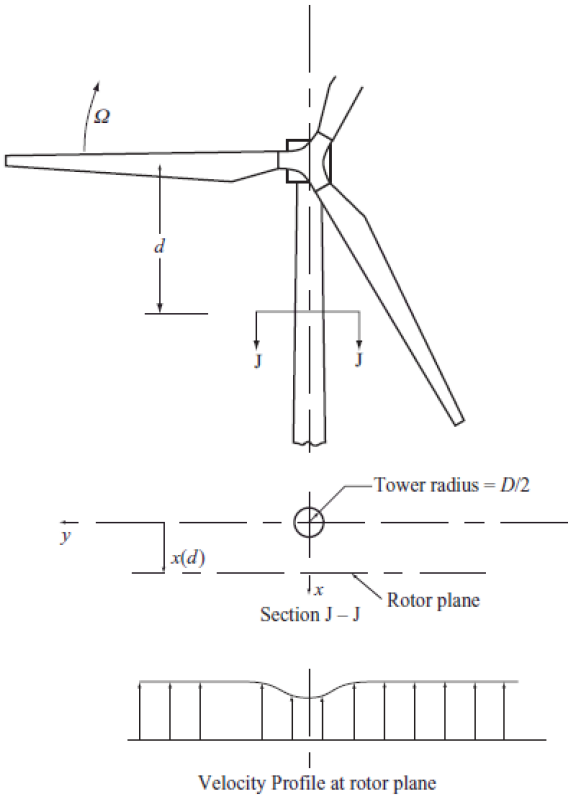

Figure 2.3: Load on an offshore wind turbine [11]

Differentiating the stream function with respect to y yield the axial velocity upwind,

$$U_x = U_\infty \left(1 - \frac{(D/2)^2(x^2 - y^2)}{(x^2 + y^2)^2}\right) \tag{2.4}$$

2.1.3 Aerodynamic Loads on Nacelle and Rotor Hub

The calculation of aerodynamic loads on non-submerged bodies such as the nacelle and rotor hub is based on the instantaneous wind and body relative velocities as in the following equation [25],

$$q = C(\alpha)v^2 \tag{2.5}$$

where

- q is the force in the degree of freedom concerned
- C is the classical wind force coefficient in the degree of freedom concerned
- α is the relative angle between the body and wind velocity
- v is the relative velocity between the body and wind

2.1.4 Aerodynamic Loads on Tower

Slender sections that are not part of the wind turbine (i.e. below the lowest blade passing point) is subjected to Morison-type quadratic drag loads [26] which is based on the relative wind velocity in local coordinates as in Figure 2.4.

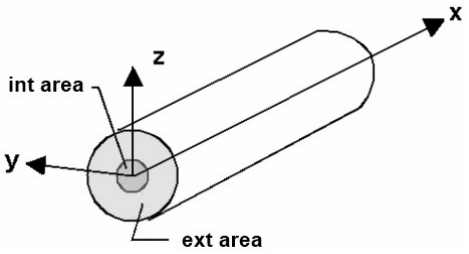

Figure 2.4: Riflex beam element local coordinate

$$u_r = u_{wind} - \dot{v} = u_{r1}i_1 + u_{r2}i_2 + u_{r3}i_3 \tag{2.6}$$

$$F_j = \frac{1}{2}\rho_{air} DLC_{dj} u_{rj}|u_{rj}| \tag{2.7}$$

where

- j is the degree of freedom
- u_r is the relative velocity
- u_{wind} is the wind velocity
- $\dot{v}$ is the tower element velocity
- F is the drag force
- D is the characteristic length
- L is the tower height
- C_d is the aerodynamic coefficient

2.2 Hydrodynamic Model

For floating offshore wind turbines, a clear distinction from fixed offshore wind turbines is the potentially large hydrodynamic loads due to larger volume and motions of the floating support structure. Interactions between floating bodies, mooring and waves is accounted for using first and second order wave theories. In this section, a breakdown of the difference sources of force relevant to the floating bodies will be discussed.

2.2.1 Body Kinetics

SIMO module performs time domain hydrodynamic forces calculation on floating bodies (SIMO bodies) using frequency dependent properties based on first and second order potential flow theories. The time domain representation of force motion can be generalized as shown in Equation 2.8. The solution method by convolution integral is presented as follows [25],

$$[m + A]\ddot{x} + C\dot{x} + B_1\dot{x} + B_2\dot{x}|\dot{x}| + K(x)x = q(x, \dot{x}, t)$$
$$q(x, \dot{x}, t) = q_{WI} + q_{WA}^{(1)} + q_{WA}^{(2)} + q_C + q_{ext} \quad (2.8)$$

where

- m is the body mass matrix
- $A(\omega) = a(\omega) + A_\infty$ is the frequency dependent added mass matrix
- A_∞ is the asymptotic added mass coefficient as $\omega \to \infty$
- K is the hydrostatic stiffness
- $C(\omega) = c(\omega) + C_\infty = c(\omega)$ is the frequency dependent potential damping
- B_1 is the linear damping matrix

- B_2 is the quadratic damping matrix
- q is the wave excitation force
- q_{WI} is the wind drag force
- $q_{WA}^{(1)}$ is the first order wave force
- $q_{WA}^{(2)}$ is the second order wave force
- q_C is the current drag force
- q_{ext} includes wave drift damping, specified forces and forces from station-keeping and coupling elements, etc

Rearranging equation 2.8

$$[m + A]\ddot{x} + C\dot{x} = q(x, \dot{x}, t) - B_1\dot{x} - B_2\dot{x}|\dot{x}| - K(x)x = f$$

and letting $x(t) = Xe^{i\omega t}$ and $f(t) = Fe^{i\omega t}$, the equation can then be represented in frequency domain as

$$-\omega^2 A_\infty X(\omega) + [i\omega a(\omega) + c(\omega)]i\omega X(\omega) = F(\omega) \tag{2.9}$$

By taking the inverse Fourier transformation the equation can be written in the final form

$$[m + A_\infty]\ddot{x} + B_1\dot{x} + B_2\dot{x}|\dot{x}| + K(x)x + \int_0^t h(t-\tau)\dot{x}(\tau)d\tau = q(x, \dot{x}, t) \tag{2.10}$$

where the retardation function $h(\tau)$ can be computed using the frequency dependent added mass or damping using equation

$$h(\tau) = \frac{2}{\pi}\int_0^\infty c(\omega)cos(\omega\tau)d\omega = -\frac{2}{\pi}\int_0^\infty \omega a(\omega)sin(\omega\tau)d\omega \tag{2.11}$$

and relationship between the frequency dependent added mass and damping can be defined according to the Kramers-Krnig relations

$$a(\omega) = -\frac{1}{\omega}\int_0^\infty h(\tau)sin(\omega\tau)d\tau \tag{2.12}$$

$$c(\omega) = -\int_0^\infty h(\tau)cos(\omega\tau)d\tau \tag{2.13}$$

2.2.2 First Order Wave Excitation Force

First order or linear wave theory allows us to superimpose waves of different frequencies, wavelengths and directions on top of each other. It implies that wave-body interactions can be studied using many regular waves and combined to obtain the resultant motion in irregular waves. The wave elevation describing an irregular wave can thus be expressed as a summation of a number of incident regular waves. Airy wave theory further describes the propagation of gravity wave as a sinusoidal wave form,

$$\zeta = \zeta_a \cos(\omega t - kx) \tag{2.14}$$

where

- ζ_a is the wave amplitude
- ω is the wave frequency
- k is the wave number associated with wave frequency through dispersion relation
- x is the horizontal coordinate of the wave propagation

Subsequently, applying non-permeable boundary conditions on the free surface and seabed allows the solving of Laplace equation for the incident wave velocity potential,

$$\phi_0 = \frac{g\zeta_a}{\omega}\frac{cosh(k(z+h))}{kh}cos(\omega t - kx) \tag{2.15}$$

where z is the vertical coordinate and h is the water depth. The first order wave excitation force can thus be obtained in terms of velocity potential by simply integrating the static and dynamic pressure over the wetted surface,

$$p = -\rho\frac{\partial\phi}{\partial t} - \rho g z \tag{2.16}$$

$$F = \int_{S_B} p\boldsymbol{n}dS \tag{2.17}$$

where ϕ is the wave velocity potential. Under the assumption of linearity, velocity potential can be superposed which allows us to further decomposed the wave interaction problem into a diffraction part and a radiation part.

Diffraction

In the diffraction problem, the body is is fixed while interacting with incoming waves. When incoming waves arrives at the body surface, diffracted waves are formed due to the body's impermeability. The load generated due to incoming wave potential is termed Froude Kriloff load while the load due diffracted wave potential is termed diffraction load and the total excitation force is the sum of these two terms as shown in,

$$F_{exc} = \underbrace{-\int_{S_B} \rho \frac{\partial \phi_0}{\partial t} \boldsymbol{n} dS}_{\text{Froude Kriloff}} \underbrace{-\int_{S_B} \rho \frac{\partial \phi_D}{\partial t} \boldsymbol{n} dS}_{\text{Diffraction}} \tag{2.18}$$

Radiation

In the radiation problem, the body is forced to oscillate in all six degree of freedoms under calm water condition with frequency ω. The oscillating body generates radiated waves with velocity potential ϕ_R and is thus subjected to reaction forces in the form of added mass, damping and restoring components. Restoring components are associated with the variation of buoyancy or other form of restoring forces due to motions and is characterized by the hydrostatic stiffness matrix. The frequency dependent added mass and damping are connected to the dynamic pressure which together will form the excitation force due to radiation,

$$F_{rad} = -\int_{S_B} \rho \frac{\partial \phi_R}{\partial t} \boldsymbol{n} dS \tag{2.19}$$

$$= \sum_{j=1}^{6} \{-A_j \ddot{\eta}_j - B_j \dot{\eta}_j\} \tag{2.20}$$

where A and B are the added mass and damping coefficients respectively.

2.2.3 Second Order Wave Excitation Force

Instead of solving the wave body interaction problem to the first order, solving it accurate to second order yield the following representation of velocity potential,

$$\phi = \phi_1 + \phi_2 \tag{2.21}$$

where the indices represent the order of solution. The wave induced load can then be obtained through direct pressure integration as described above but this time with additional second order terms in the pressure equation,

$$p = \underbrace{-\rho g z - \rho \frac{\partial \phi_1}{\partial t}}_{\text{first order}} \underbrace{-\rho \frac{\partial \phi_2}{\partial t} - \rho \frac{1}{2} \nabla \phi \cdot \nabla \phi}_{\text{second order}} \tag{2.22}$$

Due to the squared velocity term, with any 2 incident waves of frequency ω_i and ω_j propagating in x, the second order effect results in a mean drift force, a sum-frequency excitation at short period and a difference-frequency excitation at long period. Although the amplitude of the forces due to second order effect are much smaller as compared to the first order forces, In the case of a moored spar structure,

2.2.4 Current Drag

Current drag on the submerged body is calculated using Morison equation as in,

$$q_C^k(\alpha, t) = C_1^k(\alpha)|u(t)| + C_2^k(\alpha)|u(t)|^2 \tag{2.23}$$

$$|u|^2 = (v1 - \dot{x}_1)^2 + (v2 - \dot{x}_2)^2 \tag{2.24}$$

$$\alpha = arctan\frac{v_2 - \dot{x}_2}{v_1 - \dot{x}_1} \tag{2.25}$$

where

- k is the degree of freedom
- C_1, C_2 are the linear and quadratic current drag coefficient
- u is the relative velocity between the body and the current
- α is the relative angle between the velocity of the body and current
- $v1$, $v2$, $\dot{x}_1$, $\dot{x}_2$ are the current velocity components and body velocity components in the transitional degree of freedoms with respect to the body's coordinate system

2.2.5 Mooring Force

Mooring system is crucial to ensure that the platform exhibits good station keeping capability. One of the most common form of mooring is the catenary system which provides restoring forces through the suspended weight of mooring lines and resistance to the change in configuration due to vessel motions. The OC3 Hywind spar platform is moored by three catenary mooring lines 120° apart from each other. Each of these lines is connected to the platform via a so-called "crawfoot" or delta connection at one end to increase the yaw stiffness while the other end to an anchor which is attached to the seafloor.

Mooring System Modeling in SIMO

The mooring line modeling in SIMO is an extension from the model used in the mooring analysis program MIMOSA [25]. Each mooring line consists of multiple segments with different properties and clump weights. In SIMO, the analysis of mooring line involves both quasi-static analysis and dynamic analysis with dynamic analysis being performed in time domain. Force equilibrium on a mooring line element is shown in Figure 2.5 and modelled using catenary equations as follows,

$$T + \Delta T cos(\theta + \Delta\theta) - T cos\theta = 0 \tag{2.26}$$

$$T + \Delta T sin(\theta + \Delta\theta) - T sin\theta - wds = 0 \tag{2.27}$$

where

- T is the tension

- θ is the inclination
- w is the weight per unit length
- ds is the element length

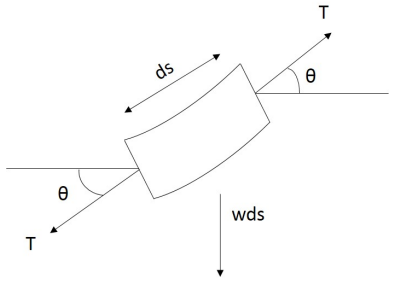

Figure 2.5: Load equilibrium on a mooring line element

Since in quasi-static analysis the transverse drag force is neglected, the mooring line tension calculated may be underestimated. SIMO has thus incorporated a model developed by Larsen and Sandvik [] to account for such dynamic effects. The model is based on the following four key assumptions,

- The only contribution to the dynamic tension comes from the tangential component of the top end motion
- The shape of the dynamic motion due to a tangential excitation is assumed to be equal to the change in static line geometry.
- Mass forces on the line are neglected.
- The elastic elongation of the line is determined quasi-statically.

2.3 Structural Model

Gravitational and inertia loads are static and dynamic loads resulting from gravity, vibration, rotation and seismic activity. These loads are taken into account in the form of SIMO bodies and slender structures. The rotating blade for example is acted on by gravity and inertial loading. Its gravity loading depends on blade azimuth and mass distribution but its inertia loading will be dependent on the wind speed fluctuations which is stochastic in nature. Structural analysis combines aerodynamic, hydrodynamic and control induced loads and solves for load responses at every time step. Different aero-hydro-elastic-servo codes carry out structural analyses differently. For example, FAST uses modal analysis, HAWC2 uses multi-body simulation while SIMA uses finite element method (FEM) analysis. The time domain load-displacement relationship is formulated in SIMA's FEM module RIFLEX based on the principle of virtual work as the following equation [26],

$$M\ddot{D} + C\dot{D} + R^{int} = R^{ext} \tag{2.28}$$

where

- $M = M^S + M^H$ is the generalized mass matrix
- M^S is the structural mass matrix
- M^H is the hydrodynamic mass matrix
- $C = C^S + C^H$ is the generalized damping matrix
- C^S is the internal structural damping matrix
- C^H is the hydrodynamic damping matrix
- R^{int} is the internal load
- R^{ext} is the external load
- D is the structural displacement

The external load R^{ext} is a combination of wind load from the blades, tower drag, and hydrodynamic loads from the floating body and mooring lines. Assuming that it is orthogonal to the eigenvectors, the local element Rayleigh structural damping, c can be modeled as a linear combination of structural mass, m and stiffness matrix, k

$$c = \alpha_1 m + \alpha_2 k \tag{2.29}$$

where α_1 and α_2 are the mass and stiffness proportional coefficients respectively.

The nonlinear time domain analysis is performed through numerical time integration of the equilibrium equation at each time step. Newton-Raphson iterations are performed at each time step in order to properly account for the effects due to model nonlinearities.

2.4 Control Model

The control algorithm plays a crucial part introducing forced excitation torque onto the rotor of wind turbine through speed and blade pitch regulation. Conventionally, the main objective of a controller is to regulate generator speed at different operating region and to ensure smooth power generation. As wind turbine increases in size, loads introduced by control algorithm increases substantially which when coupled with large motion of a floating wind turbine can introduce instability to the entire structure. This lead to the development of various load-mitigating controllers in search for the perfect balance between load reduction and power generation capabilities. As this work aims to implement and study the performance of different control algorithms, the theory leading to the implementation of each controller will be introduced and explained in Chapter 6. The performance comparison in terms of speed regulation, power fluctuation and most importantly structural load responses will be presented and discussed in Chapter 7.

2.5 Multi-body System

In MBS analysis, the gear- box is modelled as an integrated system with both rigid and flexible bodies connected through force elements. Flexible bodies such as shafts, base plate and casing for are modelled using finite element software and imported to MBS software with reduced DOFs [27]. Bearings are modelled as force elements connecting flexible bodies with linear or non-linear load-displacement relationships [28]. Gears are modelled as rigid bodies with compliance at the gear teeth. The equation of motions for the MBS dynamics can be generalized as,

$$M\ddot{X} + C\dot{X} + KX = \mathbf{F} \tag{2.30}$$

where M, C, K are the inertia, damping and stiffness matrices respectively. X is 6-DOF displacement and therefore can be represented as

$$X = \{x \quad y \quad z \quad \alpha \quad \beta \quad \gamma\}^T \tag{2.31}$$

Excitation force vector can be represented as

$$\mathbf{F} = \{F_x \quad F_y \quad F_z \quad M_x \quad M_y \quad M_z\}^T \tag{2.32}$$

To simulate the dynamic effects at a component level under a certain environment conditions, load responses as computed in global analysis specifically time series of non-torque forces and moments at the wind turbine tower top and the 6-DOF motions at the nacelle are extracted and used as inputs to MBS simulations. Figure 2.6 shows a summary of inputs to the drivetrain model in MBS simulations.

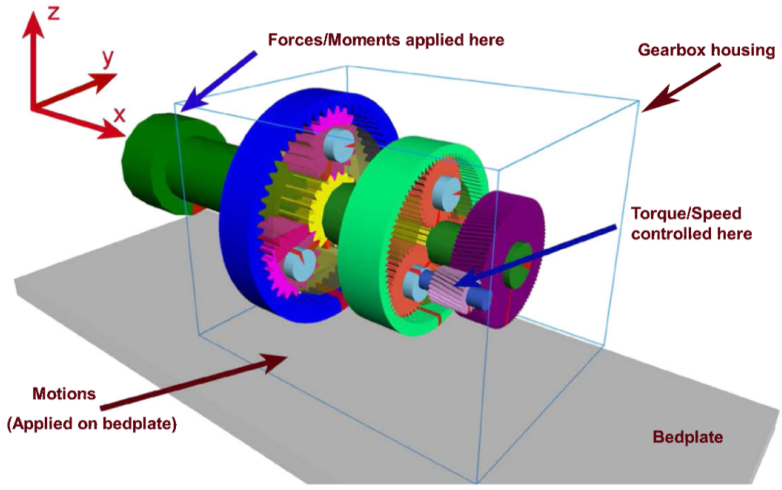

Figure 2.6: Input force vector on SIMPACK drivetrain model [7]

The input force vector is applied at the gearbox main shaft in the nacelle while the 6-DOF motion responses at the nacelle are applied through the base plate. Reference generator speed is being input through the high speed shaft and the resulting torque to be applied at the high speed shaft is computed using a PI-controller that is driven by speed error from the reference [29]. The equation for torque is as shown below,

$$T = K_P e + K_I \int_0^t e dt \tag{2.33}$$

Chapter 3

Methodology

The analysis method used in this work can be divided into three stages - model verification, global load analysis and drivetrain dynamic analysis. In general, the loads acting on an offshore wind turbine is time varying and thus time domain analysis should be carried out to fully capture the responses. The model verification global analysis of FOWT model is carried out in SIMA simulation environment which consists of four main modules - SIMO, RIFLEX, Aerodynamic Module and Control Module. Finally, drivetrain dynamic analysis is carried out in SIMPACK simulation environment to study the effect of environmental loads on drivetrain fatigue performance.

3.1 Model Verification

In order to identify some preliminary characteristics of the OC3-Hywind model as a basis for further analysis, constant wind and free decay tests are performed in SIMA and comparison is made with the system steady state behavior as documented by Jonkman et. al [6] and the floating platform hydrodynamic properties as discribed by Jonkman [5]. The characteristics of interest are,

- generator and rotor speed variation with respect to wind speed
- generator and rotor torque variation with respect to wind speed
- generator power variation with respect to wind speed
- rotor thrust variation with respect to wind speed
- rotor tip-to-speed ratio
- blade pitch variation with respect to wind speed
- platform mean offset in surge and pitch DOFs
- natural frequencies and damping coefficients of the FOWT in all surge, pitch, heave and yaw DOFs

3.1.1 Constant Wind Test

Constant wind tests on the model are performed in a calm unidirectional static wind environment without wave. This is achievable in SIMA by setting the significant wave height to a small value ($0.001m$) and the wave peak period to a high value ($20s$) to simulate a calm seastate. The model was simulated using 12 different wind speeds starting from from $4m/s$ to $24m/s$ for a duration of $800s$, long enough for the transient response to die out. The steady state values representing the wind turbine and generator characteristic are as shown in Figure 3.1 and Figure 3.2.

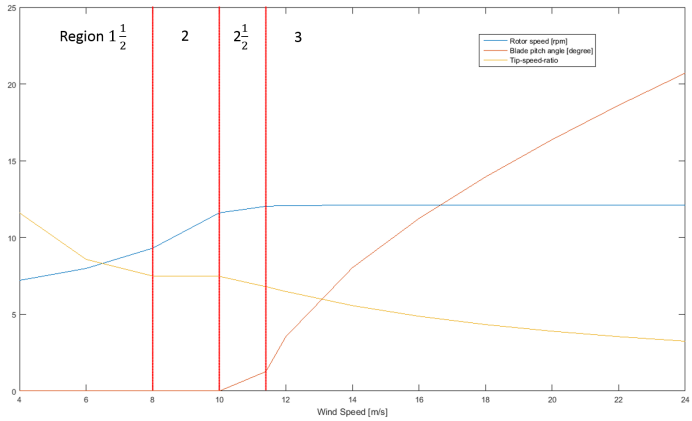

Figure 3.1: Steady wind characteristic: rotor speed, blade pitch angle and tip-speed-ratio

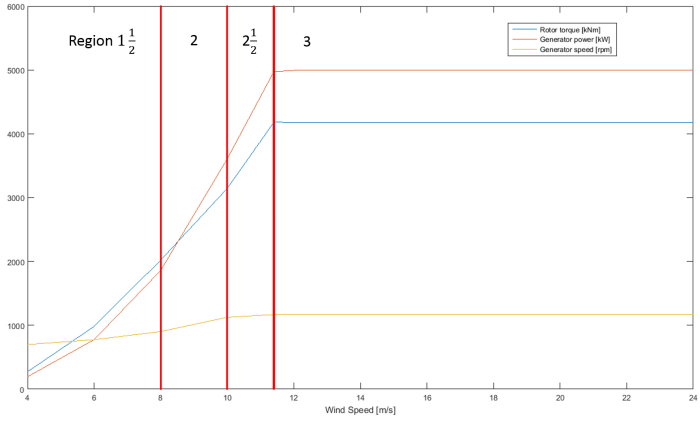

Figure 3.2: Steady wind characteristic: generator speed, generator power and generator torque

The first observation can be made by dividing the plots into different control regions

as defined by the controller. As the wind speed is increased past the transitional region $1\frac{1}{2}$, rotor and generator speeds increase linearly in region 2 with respect to wind speed in order to maintain a constant tip-to-speed ratio for optimum power capture. In the same region, generator power increases cubically with respected to wind speed while rotor torque increases quadratically. Above the rated speed of $11.4m/s$, the rotor torque is held constant through modifying the collective blade pitch angle while the generator controller maintains a constant rated generator speed. The generator power output is therefore held constant. It should be noted that similar steady state behavior can be observed even if a constant power variable pitch controller is used.

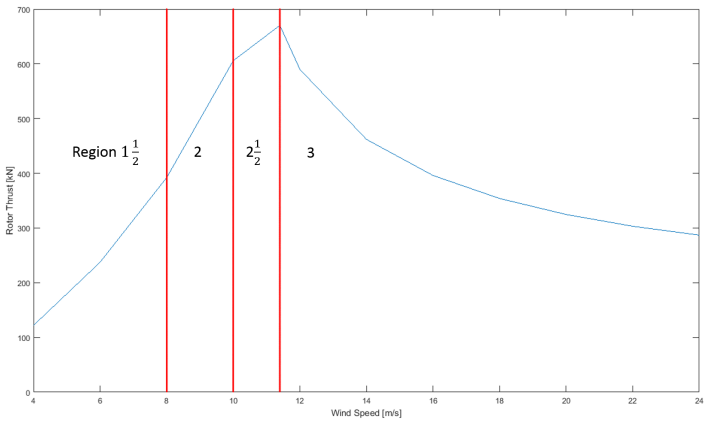

Figure 3.3: Steady wind characteristic: rotor thrust

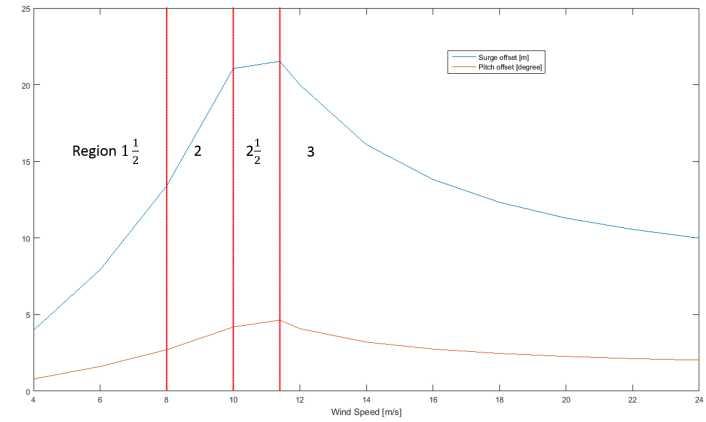

Figure 3.4: Steady wind characteristic: mean offset in surge and pitch

As presented in Figure 3.3, the thrust varies quadratically in region 2 and reaches a maximum at the rated wind speed. Above rated wind speed, the turbine blades were actuated to maintain a constant rotor torque and as the blades become more and more aerodynamic with respect to the airflow the thrust force in the wind direction decreases. Similar conclusion can be drawn as shown in Figure 3.4 with the surge and pitch mean offset reduces with respect to wind speed past rated condition. The behaviors agree with the steady state wind turbine responses as documented by Jonkman [5].

3.1.2 Free Decay Test

Decay tests are performed on parked turbine for surge, heave, pitch and yaw motions as the platform is symmetrical with respect to the x and y axes. In calm seastate, an initial ramp excitation is given followed by a constant excitation through "specified force" and "specified moment" in SIMA. The free decay motion after the forces are released is captured in time series as shown in Figure 3.5.

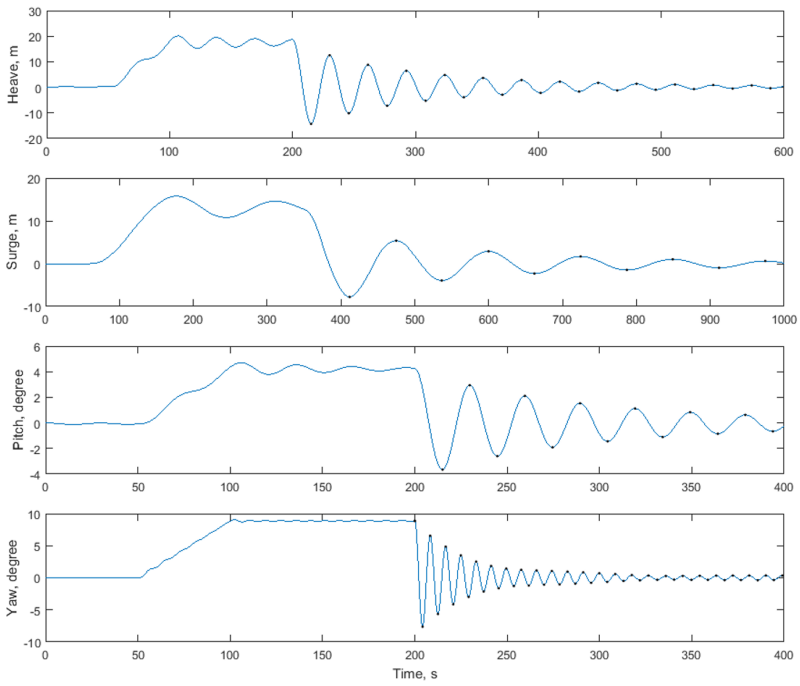

Figure 3.5: Platform decay test time series

The linear and quadratic damping coefficient is then calculated through the energy loss function. Consider a 1-DOF damped free vibration equation of motion

$$\ddot{\eta} + b_1\dot{\eta} + b_2\dot{\eta}|\dot{\eta}| + \omega_0^2\eta = 0 \tag{3.1}$$

where b_1 and b_2 are the normalized linear and quadratic damping coefficients respectively and ω_0 is the damped natural frequency. The total energy content at any time can represented by the sum of kinetic and potential energy as

$$V(t) = \frac{1}{2}\dot{\eta}^2(t) + \frac{1}{2}\omega_0^2\eta^2(t) \tag{3.2}$$

$$\frac{dV}{dt} = \dot{\eta}\ddot{\eta} + \omega_0^2\eta\dot{\eta} \tag{3.3}$$

Comparing Equations 3.1 and 3.3, the rate of change of energy can be rewritten as follows,

$$\frac{dV}{dt} = -\dot{\eta}(b_1\dot{\eta} + b_2\dot{\eta}|\dot{\eta}|) \tag{3.4}$$

The loss of energy in one cycle can be defined as

$$L(V) = \int_0^{\frac{2\pi}{\omega_0}} -\frac{dV}{dt}dt$$

$$L(V) = b_1 V + b_2\frac{16\sqrt{2}}{3}\pi V^{\frac{3}{2}} \tag{3.5}$$

Coefficients b_1 and b_2 are calculated by minimizing the overall least square between Equation 3.5 and the value of $L(V)$ obtained using cubic spline fitting of V from simulation results. The natural damped periods and damping coefficients for all motions are summarized in Table 3.1.

Motion	**Period [s]**	**linear damping,** b_1	**quadratic damping,** b_2
Surge	125.05	0.0071113	0.016597
Heave	31.17	0.018412	0.0019034
Pitch	29.82	0.015675	0.012924
Yaw	8.11	0.029792	0.016679

Table 3.1: Platform decay test properties

This is the result of additional linear damping added in the sway, surge, heave and yaw DOFs added in the model to account for the difference as compared to the responses of Hywind spar buoy calculated by Equinor [5]. The linear damping added into the spar in matrix form is,

$$B_{linear} = \begin{bmatrix} 100,000\frac{N}{m/s} & 0 & 0 & 0 & 0 & 0 \\ 0 & 100,000\frac{N}{m/s} & 0 & 0 & 0 & 0 \\ 0 & 0 & 130,000\frac{N}{m/s} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 13,000,000\frac{Nm}{rad/s} \end{bmatrix} \tag{3.6}$$

As shown, the lack of restoring force in the surge direction resulted in a high natural period. The linear damping is the lowest compared to other motions indicates that the body's wave generating capacity in the surge direction is limited due to the rounded edge of the spar. Quadratic damping on the other hand is significantly higher contributed by the second order eddy making effect of the hull and mooring lines. Heave and pitch motion are having a relatively high natural period (low compared to surge) due to the small water plane surface and hence a small restoring force and moment. The small quadratic damping in heave indicates that the second order effect is negligible in the heave direction. Yaw motion is governed mainly by the mooring connections to the fairleads providing high restoring moment and hence a low natural period.

3.2 Global Load Analysis

The purpose of global analysis is to simulate the global responses of the model under different environment conditions defined only by wind and wave. The length of the dynamic simulation is set to be $4000s$ and time step is set to be $0.005s$. This section gives a detailed description of the flow of simulation and the communications between different modules. A schematic diagram describing the interaction between modules are as shown in Figure 3.6.

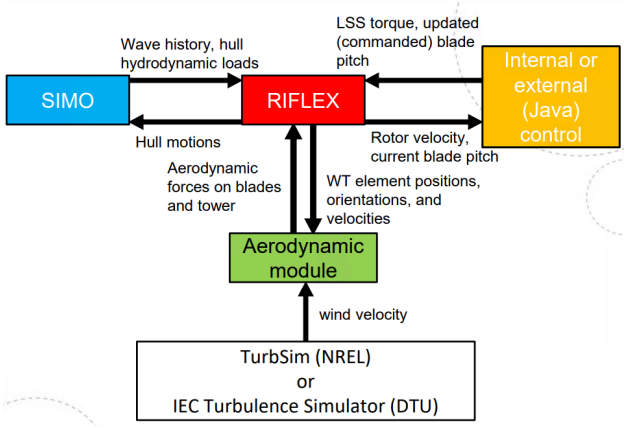

Figure 3.6: Interaction between modules in SIMA [**?**]

3.2.1 SIMO

SIMO models the rigid bodies which consists of the floating structure, hub and nacelle assembly (defined as "spar", "hubmass", and "nacelle" in SIMA respectively). It is important to note that the sum of all body masses excludes the weight of both wind turbine and mooring system and the restoring component due to aerodynamic loads on wind turbine. For the submerged part of "spar", "slender element" is used to model the Morison drag that it experiences. The frequency domain hydrostatic properties are imported from hydrodynamic analysis program WaveAnalysisMIT (WAMIT) with modifications such that the properties are accurate to only the rigid body considered and to take into account the yaw stiffness contribution from the mooring system.

3.2.2 RIFLEX

RIFLEX, the finite element solver of SIMA models flexible slender structures (i.e. tower, blades, mooring line and rotor shaft) and conducts dynamic non-linear analysis in time domain to solve for load responses and displacement to be used in the subsequent time step. Slender structures in SIMA are defined by "supernodes" which are connected through "lines". Each "supernode" defines the DOF and coordinates while each "line" is defined by its number of elements and cross sectional properties. A list of lines defined in the OC3-Hywind model is as shown in Table 3.2.

Line Name	**End Node 1**	**End Node 2**	**Length** $[m]$
shaft	sh_sn1	sh_sn2	1.0000
bl1ecc	sh_sn1	bl1e_sn2	1.5002
bl1foil	bl1f_sn1	bl1f_sn2	61.5000
bl2ecc	sh_sn1	bl2e_sn2	1.5002
bl2foil	bl2f_sn1	bl2f_sn2	61.5000
bl3ecc	sh_sn1	bl3e_sn2	1.5002
bl3foil	bl3f_sn1	bl3f_sn2	61.5000
moor_1	tp_1	anchor_1	902.2000
moor_2	tp_2	anchor_2	902.2000
moor_3	tp_3	anchor_3	902.2000
sparart	sparloc	towerlow	10.0000
tower	towerlow	towerup	77.6000

Table 3.2: Performance comparison for different controllers

For blade modelling, the cross section type "double symmetric cross section" is used on the blade elements as opposed to "generic axissymmetric pipe" because of its ability to account for aerofoil properties. The cross sections of blade elements contain material properties and aerofoil properties crucial to be used in aerodynamic load calculation which will be discuss in detailed in section 3.2.3. A shaft line which is a straight line consists of two segments (i.e. a high speed side and a low speed side) is used to model the generator shaft. A nodal connection between the two segments is given an arbritary stiffness during static analysis and the stiffness is removed during dynamic analysis when generator torque is applied at the point instead.

3.2.3 Aerodynamic Module

The aerodynamic module of SIMA solves for aerodynamic loads based on BEM method, including dynamic stall, tower shadow effect and skewed inflow correction []. As mentioned during blade modelling, the cross section aerofoil properties that are available to be used in BEM calculations are the drag coefficient, C_D, lift coefficient, C_L, and momentum coefficient, C_M. Each blade is made out of 17 elements and the properties are as shown in Table 3.3. In the table, "cyl" refers to cylindrical cross section, "du" refers to Delft University and "naca" refers to National Advisory Committee for Aeronautics.

Element no.	**Element Length, [m]**	**Chord Length [m]**	**Aerofoil**
1	2.7333	3.542	cyl1
2	2.7333	3.854	cyl1
3	2.7333	4.167	cyl2
4	4.1000	4.557	du40
5	4.1000	4.652	du35
6	4.1000	4.458	du35
7	4.1000	4.249	du30
8	4.1000	4.007	du25
9	4.1000	3.748	du25
10	4.1000	3.502	du21
11	4.1000	3.256	du21
12	4.1000	3.010	naca64
13	4.1000	2.764	naca64
14	4.1000	2.518	naca64
15	2.7333	2.313	naca64
16	2.7333	2.086	naca64
17	2.7333	1.419	naca64

Table 3.3: Performance comparison for different controllers

3.2.4 Control Module

The control module communicates with RIFLEX through a programmable java interface. The method "init()" is called at the start of the simulation to read the controller input file and to initialize control parameters. Typical parameters are such as the controller proportional, integral and differential gains and the control strategy at the region above rated wind speed. The method "step()" receives measurements such as rotor speed, blade pitch angles, blade root moments and platform motions as inputs for control strategy implementation and returns torque and blade pitch angle references as commands. The method "normalControl()" defines the control algorithm that is to be applied based on speed reference and sets the required torque and blade pitch angle references. The control ends when method "finish()" being called at the end of the simulation. The source code for the control algorithm can be found in appendix.

3.2.5 Environmental Conditions

For wave simulations, the JONSWAP 2-parameter (significant wave height, H_s and peak period, T_P) wave model is used to generate wave history with time step of $\Delta t = 0.1s$ and frequency resolution of $\Delta\omega = 9.59 \times 10^{-4} rad/s$. For constant wind simulations, 2D wind time series with constant axial velocity is generated. For turbulent wind simulations, TurbSim [30] developed by NREL is used to generate 3D-wind time series (characterized by mean wind speed, U and turbulence intensity, I) with $\Delta t = 0.05s$ and grid point matrix dimension of 32×32 based on Kaimal [31] power spectral. The IEC Normal Turbulence Model (NTM) [31] is used to define the turbulence intensities for different wind speeds in this work as shown in Equation **??**.

$$
\begin{aligned}
I &= \frac{\sigma_1}{V_{hub}} = \frac{I_{ref}}{V_{hub}}(0.75V_{hub} + b) \\
b &= 5.6m/s
\end{aligned}
\tag{3.7}
$$

where I_{ref} is 0.12 for Category C wind turbine class.

Table 3.4 shows a summary of the environmental conditions used in this work. For each environmental condition, five different random wave and wind seed combinations are used to test the robustness of the controller that their performance is stable in any given random environmental condition. The global responses calculated using one of the five seeds is then randomly selected to be used in the subsequent drivetrain dynamic analysis.

	EC4	**EC5**	**EC6**
Significant wave height, $H_S[m]$	5.0	4.0	5.5
Peak period, $T_P[s]$	12.0	10.0	14.0
Mean wind speed, $U[m/s]$	12.0	14.0	20.0
Turbulence intensity, $I[-]$	0.15	0.14	0.12

Table 3.4: Performance comparison for different controllers

3.3 Drivetrain Dynamic Analysis

The MBS model of the 5MW reference gearbox used in this work was developed by Nejad et. al. [7] using SIMPACK, a multipurpose, multibody simulation code that can be used to simulate gearbox dynamics [32]. The gear tooth contact is modeled by the force element FE225 in SIMPACK and bearings are modeled with linear force-deflection relation. The proportional and integral gains K_p and K_I for generator torque calculation are chosen as $K_P = 2200$ and $K_I = 220$. The time step of the simulation is $dt = 0.005s$. The first $200s$ of the transient response with effects due to turbine start up is discarded prior to being imported into SIMPACK. Figure 3.7 shows the coordinate transformation connecting the simulation tools and Equation 3.8 to 3.13 summarize the mapping of outputs to inputs.

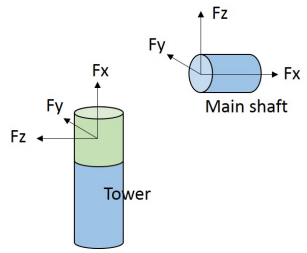

Figure 3.7: Coordinate transformation from tower top to main shaft

$$F_z = F_{TowerTop_x} - (-240 \times 9.81) \tag{3.8}$$
$$F_y = F_{TowerTop_y} \tag{3.9}$$
$$F_x = -F_{TowerTop_z} \tag{3.10}$$
$$M_x = M_{TowerTop_z} \tag{3.11}$$
$$M_y = M_{TowerTop_y} - 240 \times 9.81 \times 1.9 \tag{3.12}$$
$$M_z = M_{TowerTop_{torsion}} \times \frac{1.9}{5} \tag{3.13}$$

where the units are in kN for forces and kNm for moments. $240T$ represents the mass of nacelle with a center of gravity $1.9m$ above the tower top.

3.4 Performance Comparison

Several parameters are chosen to quantify the performance of different load mitigating control strategies and they will all be presented as a comparison to the baseline controller. The performance parameters will be presented in percentage change in comparison to the baseline as summarized in Table 3.5. The rest of the section will describe the calculation method for each parameter.

Component	Parameter	Description
Tower Base	$D_{TowerBase}$	Tower base 1-hr fatigue damage
Gear	D_{gear}	Gear root 1-hr fatigue damage
Bearing	$D_{bearing}$	Bearing 1-hr fatigue damage
Blade Root	D_{blade}	Blade root 1-hr fatigue damage
Pitch Actuator	ADC	Pitch actuator duty cycle
Power quality	σ, μ	Standard deviation and mean value of power output

Table 3.5: Performance comparison for different controllers

3.4.1 Tower Base Fatigue Damage

The short term fatigue damaged estimation of the tower base is carried out using 1-hour of simulation data from global analysis. The axial stress at the tower base was calculated using axial force and bending moments output from RIFLEX in local element coordinate as shown in Figure 3.7

$$\sigma_x = \frac{N_x}{A} + \frac{M_y}{I_y} r sin\theta + \frac{M_z}{I_z} r cos\theta \tag{3.14}$$

where

- N_x is the axial force
- M_y and M_z are the bending moment in the fore-aft and side-side directions respectively
- I_y and I_z are the area moment of inertia of the tower cross section in the fore-aft and side-side directions respectively

The fatigue damage accumulation was calculated based on DNV-RP-C203 [33],

$$D_{TowerBase} = \frac{1}{\overline{a}} \sum_{i=1}^{j} n_i (\Delta\sigma_i)^m \tag{3.15}$$

where

- D is the accumulated fatigue damage
- $\overline{a}$ is the intercept of the design S-N curve with the log N axis
- m is the negative inverse slope of the S-N curve
- j is the number of stress blocks
- n_i is the number of stress cycles in stress block i

and the S-N curve used is according to DNV-RP-C203 Table 2-1 [33] represented by equation

$$logN = log\bar{a} - mlog\left[\Delta\sigma\left[\frac{t}{t_{ref}}\right]^{k}\right] \tag{3.16}$$

where

- N is the predicted number of cycles to failure for stress range $\Delta\sigma$
- t is the thickness through which a crack will most likely grow
- t_{ref} is the reference thickness equal 25 mm for welded connections other than tubular joints
- k is the thickness exponent on fatigue strength

The parameters used are summarized in Table 3.6.

N<10^7		N>10^7		Fatigue limit at 10^7 cycles	k	t_{ref}
m	$log\bar{a}$	m	$log\bar{a}$			
3.0	12.164	5.0	15.606	52.63 MPa	0.2	25mm

Table 3.6: Parameters for fatigue damage estimation

3.4.2 Bearing and Gear Fatigue Damage

The dynamic forces on the MBS model are then post-processed and the 1-hr fatigue damage on the gears and bearing is calculated. For the bearings, the desired life is calculated based on IEC 61400-4 [] and expressed as,

$$L = \left(\frac{C}{P}\right)^{a} \tag{3.17}$$

where

- L is the desired life
- C is the bearing basic load rating
- P is the dynamic equivalent radial load
- a is a bearing constant with $a = 3$ for ball bearings and $a = \frac{10}{3}$ for roller type bearings

Dynamic equivalent radial load is calculated according to ISO 281-2007 []

$$P = XF_r + YF_a \tag{3.18}$$

- X and Y are dynamic radial load factors
- F_r is the bearing radial load
- F_a is the bearing axial load

The load range cycle is calculated by load duration distribution (LDD) method. This conservative approach as described in ISO 6336-6 [] suggests dividing load into bins of load range and relate the bin with the maximum load within. In this work, P is divided into 100 bins each characterized by its maximum load. The bearing fatigue damage can then be estimated through

$$D_{bearing} = \sum_i \frac{l_i}{L_i} = \frac{1}{C^a} \sum_i l_i P_i^a \tag{3.19}$$

where l_i is the number of cycles at load range P_i while L_i is the number of cycles to failure at load range P_i as calculated from equation 3.17. For estimation of gear fatigue damage, the gear tooth bending stress is required. The method of calculating permissible stress as stated in ISO 6336-3 [34] is used to estimate gear tooth bending stress.

$$\sigma_{FP} = \frac{F_t}{b_{m_n}} Y_F Y_S Y_\beta Y_{DT} K_A K_V K_{F\beta} K_{F\alpha} \tag{3.20}$$

where σ_{FP} is the gear tooth permissible bending stress, b is the gear face width and m_n is the normal module. Constant Ks and Ys are geometry dependent constants as can be referred to in ISO 6336-3. The counting of stress cycles are through LDD method similar to the counting of load cycles of bearing load. In this work, the gears are considered to be made of 16MnCr5 case-hardened which exhibits the S-N curve properties of $m = 6.225$ and $K_c = 10^{24.744}$ as shown below,

$$N_{ci} = K_c \sigma_{FPi}^{-m} \tag{3.21}$$

where N_{ci} is the characteristic cycle to failure at stress $sigma_{FPi}$. The 1-hr gear tooth damage can be calculate by direct method,

$$D_{gear} = \sum_i \frac{n_i}{N_{ci}} = \frac{1}{K_c} \sum_i n_i \sigma_{FPi}^m \tag{3.22}$$

For comparison purposes, the fatigue damage in this work is presented as change in percentage when different controllers are used as in,

$$\chi = \frac{D_{c2} - D_{c1}}{D_{c1}} \times 100\% \tag{3.23}$$

with negative value χ indicates a decrease in fatigue damage with the use of controller type 2 ($c2$) as compared to controller type 1 ($c1$). Since only relative values as compared to the baseline controller are of interest, the constants can be omitted which means that for bearing damage, only the load ranges, P_i^a and their corresponding cycles, l_i are of interest. Similarly for gear tooth damage, only the stress ranges, σ_{FPi}^m and their corresponding cycles, n_i are of interest.

3.4.3 Blade Root Fatigue Damage

Due to the highly complex blade geometry and material properties, the estimation of blade root fatigue damage will be based on the simplified load spectrum method as proposed by Freebury et. al. [35]. Freebury proposed a method that remains in load domain without the need for conversion to stresses. As a result an M-N (moment versus allowable cycles to failure) curve instead of an S-N curve as in conventional methods. The M-N relationship is defined by

$$M_{ai} = M_u N_i^{-\frac{1}{p}} \tag{3.24}$$

where

- M_{ai} is the moment amplitude of cycle i
- M_u is the characteristic maximum bending moment of the blade
- N_i is the allowable cycle to failure subjected to moment M_{ai}
- p is the slope of the M-N curve

The moment cycle, M_{ai} is obtain through rainflow counting method as shown by in [35]. The 1-hr fatigue damage on a blade due to moment in one direction can then be estimated through

$$D_{blade} = \sum_i \frac{n_{bi}}{N_i} = \frac{1}{M_u^p} \sum_i n_{bi} M_{ai}^p \tag{3.25}$$

Similar to the damage estimation for gears and bearings, for comparison purposes the constant M_u is neglected.

3.4.4 Pitch Actuator Duty Cycle Calculation

Since the blade pitch actuator is one of the common fault on during the operation of a wind turbine, it is also the intention to reduce the its activity. According to Bottasso et. al. [36], one of of quantifying the use of pitch actuator over time is through the equation

$$ADC = \frac{1}{T} \int_0^T \frac{|\dot{\theta}(t)|}{\dot{\theta}_{max}} dt \tag{3.26}$$

where

- ADC is the actuator duty cycle
- T is the operating period
- $\dot{\beta}$ is the actuator pitch rate

3.4.5 Power and Speed

Finally, the power and speed excursion is compared using the mean and standard deviation of their respective 1-hr time series.

Chapter 4

Spar Typed Wind Turbine

Spar typed platforms are widely used in the oil and gas industry mainly to accommodate deep water production facilities. It consists of a large hollow caisson ballasted to float vertically with a deck super structure installed above. Helical strakes can be installed around the caisson and are found to be very effective in reducing vortex induced vibration (VIV) that is a common problem for slender structures [37].

4.1 Catenary Moored Spar Wind Turbine Characteristic

Spar supported wind turbines are characterized by the deep draft spar buoy, small water plane area and the low center of mass (CM) with respect to the center of buoyancy (CB) due to heavy ballast. Such features result in a smaller heave motion as well as a high restoring moment which is the main source for roll and pitch stabilization. The catenary mooring system contributes mainly to the restoring forces in surge and sway directions to keep the supported wind turbine in position for optimal power capture. As for yaw motion, an alternative delta mooring connection has also been employed in the Hywind concept [18] to provide high stiffness in the yaw direction and reduces the yaw motion induced mainly by wind. This combination of characteristic makes the spar typed configuration a very attractive solution for deep water wind energy production. A schematic of a typical spar supported wind turbine with delta mooring connections is shown in Figure (4.1).

Simplifications have been made to the OC3-Hywind model to simplify the analysis as follows,

- Delta connections are eliminated. As such a yaw stiffness of 98,340,000 Nm/rad have to be compensated in the hydrostatic stiffness matrix.
- Segments of lines are replaced with elements of homogeneous physical properties and weight based on the original length of the segments
- The mooring system damping is neglected.

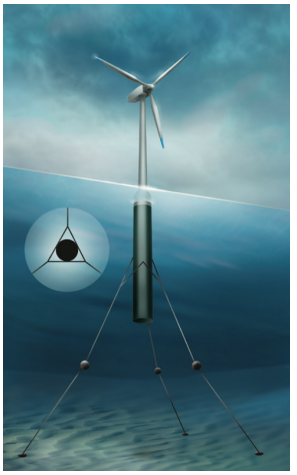

Figure 4.1: Moored spar wind turbine with delta mooring connections (CeSOS Annual Report 2009)[12]

4.2 OC3-Hywind Spar Wind Turbine

The model that will be used for the subsequent analysis is the OC3-Hywind model. It was developed by the Offshore Code Comparison Collaboration (OC3) task force imitating the Hywind spar platform originally conceptualize by Statoil to support a 5MW turbine. The updated model supports a 5MW NREL wind turbine with modified controller and support structure properties while keeping the same aerodynamics and structural properties of the turbine[5]. The support structure consist of a deep draft slender spar buoy moored by catenary mooring system as shown in Figure (4.2).

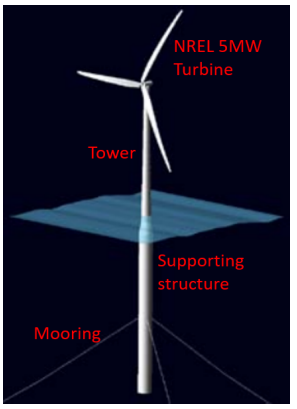

Figure 4.2: OC3-hywind [5]

4.2.1 Spar Properties

The properties of the OC3-Hywind spar are summarized in Table (4.1).

Depth to Platform Base Below SWL (Total Draft)	120 m
Elevation to Platform Top (Tower Base) Above SWL	10 m
Depth to Top of Taper Below SWL	4 m
Depth to Bottom of Taper Below SWL	12 m
Platform Diameter Above Taper	6.5 m
Platform Diameter Below Taper	9.4 m
Platform Mass, Including Ballast	7 466 330 kg
CM Location Below SWL Along Platform Centerline	89.9155 m
Platform Roll Inertia about CM	4 229 230 000 kgm^2
Platform Pitch Inertia about CM	4 229 230 000 kgm^2
Platform Yaw Inertia about Platform Centerline	164 230 000 kgm^2
Number of Mooring Lines	3
Angle Between Adjacent Lines	120°
Depth to Anchors Below SWL (Water Depth)	320 m
Depth to Fairleads Below SWL	70 m
Radius to Anchors from Platform Centerline	853.87 m
Radius to Fairleads from Platform Centerline	5.2 m
Unstretched Mooring Line Length	902.2 m
Mooring Line Diameter	0.09 m
Equivalent Mooring Line Mass Density	77.7066 kg/m
Equivalent Mooring Line Weight in Water	698.094 N/m
Equivalent Mooring Line Extensional Stiffness	384 243 000 N
Additional Yaw Spring Stiffness	98 340 000 Nm/rad

Table 4.1: OC3-Hywind spar properties [5]

4.2.2 Controller Properties

The properties of the controller are summarized in Table (4.2).

Corner Frequency of Generator-Speed Low-Pass Filter	0.25 Hz
Peak Power Coefficient	0.482
Tip-Speed Ratio at Peak Power Coefficient	7.55
Rotor-Collective Blade Pitch Angle at Peak Power Coefficient	0°
Generator-Torque Constant in Region 2	0.0255764 Nm/rpm^2
Rated Mechanical Power	5.296610 MW
Rated Generator Torque	43 093.55 Nm
Transitional Generator Speed between Regions 1 and $1\frac{1}{2}$	670 rpm
Transitional Generator Speed between Regions $1\frac{1}{2}$ and 2	871 rpm
Transitional Generator Speed between Regions $2\frac{1}{2}$ and 3	1 161.963 rpm
Generator Slip Percentage in Region $2\frac{1}{2}$	10 %
Minimum Blade Pitch for Ensuring Region 3 Torque	1°
Maximum Generator Torque	47 402.91 Nm
Maximum Generator Torque Rate	15 000 Nm/s
Proportional Gain at Minimum Blade-Pitch Setting	0.01882681 s
Integral Gain at Minimum Blade-Pitch Setting	0.008068634
blade-pitch Angle at which the Rotor Power has Doubled	6.302336°
Minimum Blade-Pitch Setting	0°
Maximum Blade-Pitch Setting	90°
Maximum Absolute Blade Pitch Rate	$8^\circ/s$

Table 4.2: OC3-Hywind controller properties [6]

4.2.3 Drivetrain Properties

The drivetrain model presented in this work is as described by Nejad et. al. [7]. The properties of the drivetrain are summarized in Table (4.3).

Type	Two planetary and one parallel
Rated Input Shaft Speed	12.1 rpm
Rated Generator Speed	1165.9 rpm
First Stage Ratio	3.947 : 1
Second Stage Ratio	6.167 : 1
Third Stage Ratio	3.958 : 1
Rated Input Shaft Torque	3946kNm
Rated Generator Shaft Torque	40.953 kNm
Total Dry Mass	53 000 kg
Service life	20 $year$

Table 4.3: OC3-Hywind drivetrain properties [7]

Chapter 5

Wind Turbine Control

5.1 The Control Induced Problem

In order to maintain optimal power capture in different operating conditions and for the safe operations of wind turbine especially at high wind speed, some form of control strategy have to be imposed to oversee the operations. For large commercial wind turbines, a combination of generator torque and blade pitch controller has been widely adopted by the industry and proven to be effective in achieving the control objectives. However, while it successfully ensure good power quality for the case of fixed bottom wind turbines, the use of conventional fixed turbine controller on floating offshore wind turbines (FOWT) opened up a different set of issues due to the non-fixed nature of a FOWT. On top of the existing wave induced motions, the coupling between blade pitch control system and platform dynamics further exacerbates the response especially in the pitch direction. The phenomenon is identified by Nielsen et al [18] as particularly destructive for FOWTs as the low frequency wind energy usually concentrates around the pitch natural frequencies of FOWTs. The problem can be better elaborated using a platform pitching scenario described in Figure (5.1).

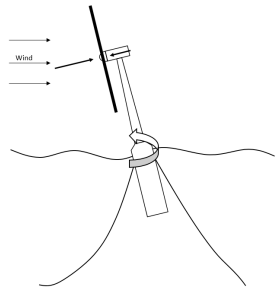

Figure 5.1: Controller induced response on a spar wind turbine during pitching

As a platform pitches upwind at above rated wind speed, the wind turbine experiences a higher relative wind speed as seen from the nacelle causing an increase in thrust force. To counter this increase in thrust the blade pitch controller that is programmed to maintain a constant power (or torque depending on the control mode) is then activated to reduce the pitch angles of the blades. When interacting with the upwind pitching motion, such reduction in rotor thrust causes the platform to take a longer time to restore to the neutral position and in the some cases even making it unstable if the onshore turbine controller gain is not tuned to take into account the induced negative damping [8].

In the following chapters, the implementation of three different control strategies will be discussed and compared with the detuned FOWT blade pitch and generator torque controller detailed by Jonkman et al [6] that is without any augmentation of rotor reference speed.

5.2 Baseline Controller

In order to provide the basis for the investigation of effects different control strategies have on the NREL 5MW wind turbine, the baseline variable speed, variable pitch configuration proposed by Jonkman et al [6] is used as a basis of comparison and will in the subsequent chapters be referred to as the baseline controller. The design consists of two independent control systems: a generator torque controller and a rotor collective blade pitch controller.

5.2.1 Generator Torque Controller

The generator torque controller is responsible of meeting the predefined speed requirement at different control regions through alternating its applied torque. Region 1 is the period when wind is being used to accelerate the rotor and no power is being extracted from this region. Hence the generator torque is set to zero at this region. Region 2 corresponds to the region of optimizing power capture through maintaining an optimum blade tip to wind speed ratio which corresponds to a torque constant of $0.0255764 Nm/rpm^2$ [6]. The torque is proportional to square of generator speed in region 2. In region 3, the generator will maintain a constant power output. It is noteworthy to point out that the simulations carried out throughout this work are based rather on a constant torque setting to minimized drivetrain load and pitch activity [19]. A generator torque-speed response curve is as shown in Figure 5.2.

5.2.2 Variable Pitch Controller

The variable pitch controller mainly works above rated wind speed maintaining constant rotor speed as wind speed increases. The major part of the analysis throughout this report will be carried out based on above rated wind condition where the variable pitch controller works in. As such a more detailed description will be presented here.

By defining the relationship of a small pitch perturbation about any instantaneous operating point $\Delta\theta$ as an output of a PID controller based on the rotor speed perturbation

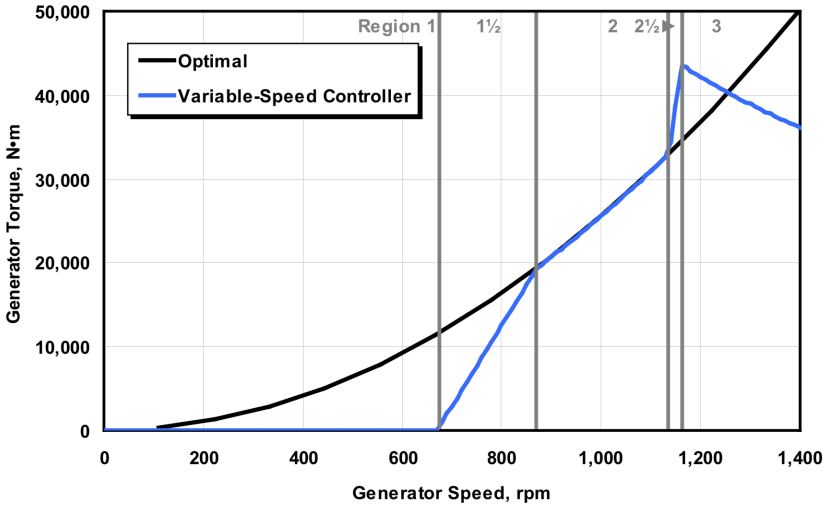

Figure 5.2: Generator torque controller: torque-speed response curve

$\Delta\Omega$ as follows,

$$\Delta\theta = K_P N_{Gear}\Delta\Omega + K_I \int_0^t N_{Gear}\Delta\Omega dt + K_D N_{Gear}\Delta\dot{\Omega} \tag{5.1}$$

the pitch controlled can be simplified to a single degree of freedom (SDOF) second order system as follows,

$$\left\{I_{drivetrain} + \frac{1}{\Omega_0}\left[\frac{\partial P}{\partial\theta}\right] N_{Gear}K_D\right\}\ddot{\varphi} + \left\{\frac{1}{\Omega_0}\left[\frac{\partial P}{\partial\theta}\right] N_{Gear}K_P - \frac{P_0}{\Omega_0^2}\right\}\dot{\varphi}... \\ ... + \left\{\frac{1}{\Omega_0}\left[\frac{\partial P}{\partial\theta}\right] N_{Gear}K_I\right\}\varphi = 0 \tag{5.2}$$

where

- $\dot{\varphi} = \Delta\Omega$
- Ω_0 is the rated rotor speed
- P_0 is the rated mechanical
- $I_{drivetrain}$ is the inertia of the drivetrain on the low speed shaft
- N_{gear} is the gear ratio
- $\frac{\partial P}{\partial\theta}$ is the sensitivity of aerodynamic power to the blade pitch angle
- K_P, K_I and K_D are the proportional, integral and derivative gain of the controller respectively

With the differential gain taken as zero, the error dynamics can be seen as a second order system with natural frequency, ω_n and damping ratio, ζ

$$\omega_n = \sqrt{\frac{I_{drivetrain}}{\frac{1}{\Omega_0}\left[\frac{\partial P}{\partial \theta}\right] N_{Gear} K_I}} \tag{5.3}$$

$$\zeta = \frac{\frac{1}{\Omega_0}\left[\frac{\partial P}{\partial \theta}\right] N_{Gear} K_P - \frac{P_0}{\Omega_0^2}}{2 I_{drivetrain} \omega_n} \tag{5.4}$$

which directly relates gains, natural frequency and system damping. As the blade pitch sensitivity $\frac{\partial P}{\partial \theta}$ changes considerably with wind speed in region 3, controller gains will have to vary according to wind speed. In region 3, blade pitch sensitivity is found to be vary linearly with blade pitch angle as shown in Figure 5.3 according to the regression analysis performed by Jonkman et al [6].

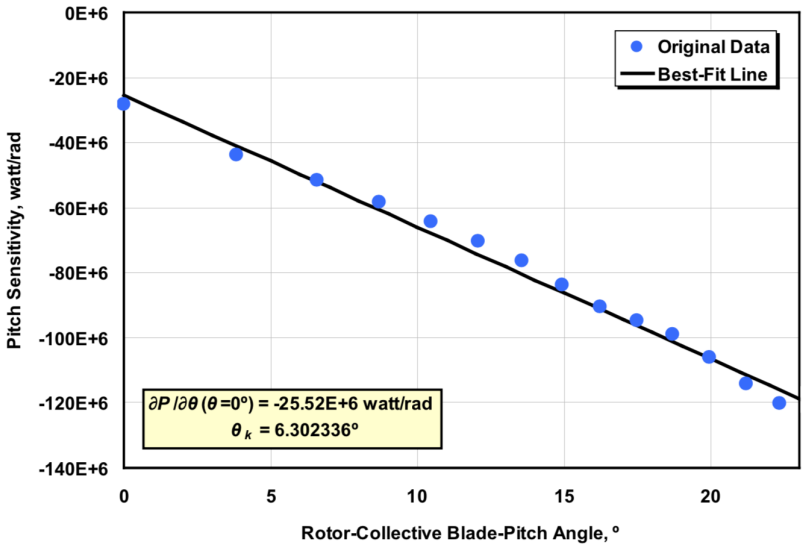

Figure 5.3: Blade pitch sensitivity with blade pitch angle [6]

Using the linear relation, controller gains can be scheduled according to

$$K_P(\theta) = \frac{2 I_{drivetrain} \Omega_0 \zeta \omega_n}{N_{Gear}\left[-\frac{\partial P}{\partial \theta}|_{\theta=0}\right]} GK(\theta) \tag{5.5}$$

$$K_I(\theta) = \frac{I_{drivetrain} \Omega_0 \omega_n^2}{N_{Gear}\left[-\frac{\partial P}{\partial \theta}|_{\theta=0}\right]} GK(\theta) \tag{5.6}$$

$$GK(\theta) = \frac{1}{1 + \frac{\theta}{\theta_k}} \tag{5.7}$$

where $GK(\theta)$ is the gain scheduling factor and θ_k is the blade pitch angle where the blade pitch sensitivity double its value ie. $\frac{\partial P}{\partial \theta}|_{\theta=\theta_k} = 2\frac{\partial P}{\partial \theta}|_{\theta=0}$. A flow diagram of the baseline controller is as shown in FIgure 5.4

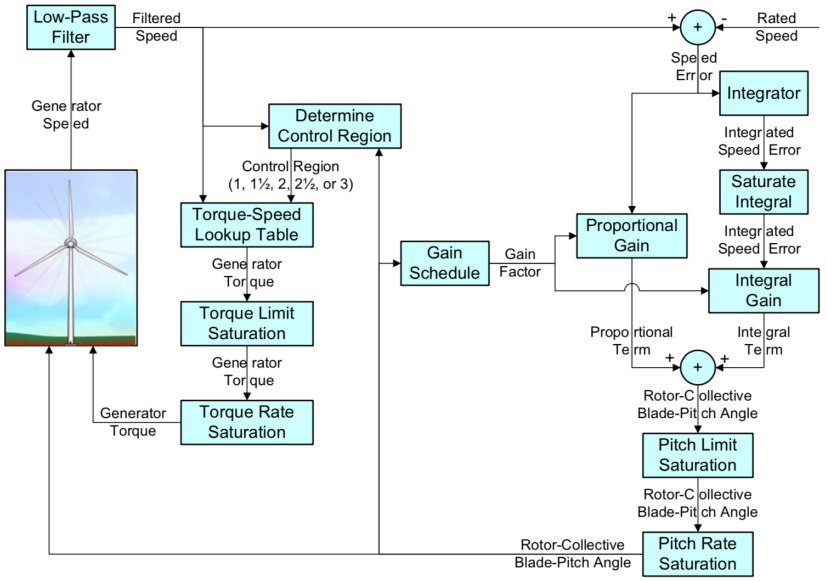

Figure 5.4: Illustrated flow diagram of baseline controller [6]

As shown in Equation 5.2, the baseline controller introduced a negative damping $-\frac{P_0}{\Omega_0^2}$ in the process of regulating rotor speed. This negative damping have to be compensated by sufficient proportional gain for the system to be stable. Since it only aims to maintain a constant rotor speed, it is clear that the baseline controller is not an optimal solution in terms of minimizing platform motions. Instead of constant reference, the three alternative methods described below aim to augment the reference rotor speed fed into the baseline controller. Consider the baseline controller's feedback loop is driven by the following error term

$$\Omega_e = \Omega_{ref} - \Omega_r \tag{5.8}$$

5.3 Baseline Controller with Active Damping

As proposed by Lackner [1], on top of the baseline controller's rotor speed error feedback loop, platform pitch velocity is being injected to the control loop to modify the generator reference. By doing this, instead of maintaining a constant generator reference speed, it is now directly related to platform pitching motion which can be used a mean to limit the platform motion.

5.3.1 Control Theory

The application is very straight forward and it can be described by first by considering the simplified version of block diagram for the baseline controller omitting gain scheduling

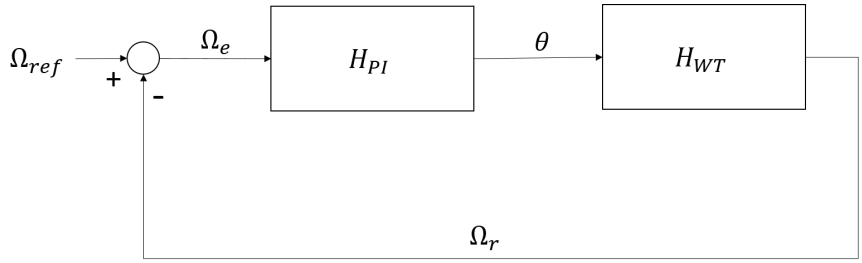

Figure 5.5: Simplified block diagram of baseline controller

Where H_{PI} is the transfer function of the variable pitch controller from and H_{WT} is the control plant of the wind turbine. In a nutshell, baseline controller works in a feedback loop that aims to drive the error term Ω_e to zero which can be represented by close loop transfer function $T(s)$ in s domain

$$T(s) = \frac{\Omega_r}{\Omega_{ref}}(s) = \frac{H_{PI}(s)H_{WT}(s)}{1 + H_{PI}(s)H_{WT}(s)} \tag{5.9}$$

According to final value theorem,

$$\lim_{t\to\infty} \Omega(t) = \lim_{s\to 0} s\Omega_r(s) = \lim_{s\to 0} sT(s)\Omega_{ref}(s) \tag{5.10}$$

Let $\Omega_{ref}(t) = \Omega_0$ at equilibrium, $\Omega_{ref}(s) = \frac{\Omega_0}{s}$

$$\lim_{s\to 0} sT(s)\Omega_{ref}(s) = \lim_{s\to 0} T(s)\Omega_0 \tag{5.11}$$

It is shown that for $\Omega_r(t)$ to track according to Ω_{ref}, $T(s)$ have to be 1 and thus H_{PI} have to be tuned in such a way that

$$|H_{PI}(j\omega)H_{WT}(j\omega)| \gg 1 \tag{5.12}$$

within a desired bandwidth of the control system. As mentioned by Skaare and Nielsen, injecting an additional active damping term after H_{PI} would result in the active damping term being suppressed within the desired bandwidth which render the injection term ineffective [8]. As such it is proposed that the injection term being injected prior to transfer function H_{PI}

The transfer function can be expressed as

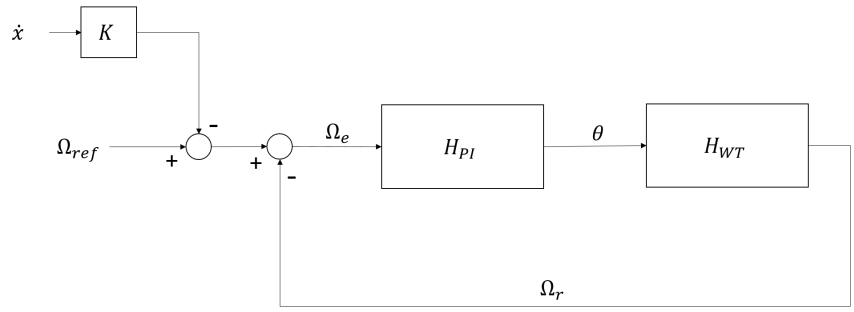

Figure 5.6: Simplified block diagram of baseline controller with active damping

$$\Omega_r(s) = \Omega_{ref}\frac{H_{PI}(s)H_{WT}(s)}{1 + H_{PI}(s)H_{WT}(s)} - K_{ad}(s)\dot{x}(s)\frac{H_{PI}(s)H_{WT}(s)}{1 + H_{PI}(s)H_{WT}(s)}$$
$$\Omega_r(s) = \Omega_{ref,ad}\frac{H_{PI}(s)H_{WT}(s)}{1 + H_{PI}(s)H_{WT}(s)} \tag{5.13}$$

where $\dot{x}$ is in this case the surge velocity as opposed to pitch velocity as used by Lackner and $K_{ad}(s)$ is a positive proportional gain of the injection term. With the same tuning of H_{PI}, the transfer function can then be re-written as

$$\Omega_r(s) = \Omega_{ref}(s) - K_{ad}(s)\dot{x}(s) \tag{5.14}$$
$$\Omega_e = \Omega_{ref,ad} - \Omega_r \tag{5.15}$$

and in time domain

$$\Omega_r(t) = \Omega_{ref}(t) - K\dot{x}(t) = \Omega_0 - K_{ad}\dot{x}(t) \tag{5.16}$$
$$\tag{5.17}$$

For implementation, Equation 5.18 can be discretized using Euler forward as

$$\Omega_{k+1} = \Omega_0 - K_{ad}\dot{x}_k \tag{5.18}$$

where subscript k represents the k-th time step. Note that the reference rotor speed is no longer a constant but a variable which is dependent on the platform surge velocity and the negative sign means that a positive surge velocity (upwind) would reduce the reference rotor speed resulting a smaller thrust force upwind. The expression in Equation 5.18 resulted in an rather intuitive method that will be directly implemented in the controller java code.

5.3.2 Controller Tuning

The only parameter to tune for this controller is the proportional gain K_{ad}. According to Lacker, 3 gain values that were being chose empirically are $0.0125s/deg$, $0.025s/deg$ and $0.0375s/deg$ [1]. Since the numerical value of surge motion amplitude is approximately 10 times the pitch motion amplitude in degree, a first estimation is to assume gain values of 10 times greater ie. $0.125s/m$, $0.25s/m$ and $0.375s/m$. The gain is chosen as $0.375s/m$ in this work.

5.4 Energy Shaping Controller

This method is proposed by Pedersen which is based on the conservation of energy within the rotor system [2]. Similar to the previous method, it proposed an injection term to update the rotor reference speed only this time rather than a proportional gain a slightly more sophisticated term is introduced based on first principles. The energy shaping controller will in subsequent chapters be referred to as ES controller.

5.4.1 Control Theory

Consider the rate change of kinetic energy of a 1-DOF drivetrain system

$$\begin{aligned} \dot{\kappa} &= F(v_{air} - \dot{x}) - E \\ \dot{\kappa} + E - Fv_{air} + F\dot{x} &= 0 \end{aligned} \tag{5.19}$$

where F is the thrust force, v_{air} is the wind speed and E is the net power of the drivetrain. In equilibrium, $\dot{\kappa} = 0$ and

$$E = E_0 = F(v_{air} - \dot{x}) \tag{5.20}$$

However, if E is allowed to vary around the point of equilibrium as a function of Ω and $\dot{\kappa} = J\ddot{\Omega}\Omega$, the equation can be re-written as

$$J\ddot{\Omega}\Omega + E(\Omega) - E(\Omega_0) + F\dot{x} = 0 \tag{5.21}$$

By comparing Equations 5.19 and 5.21, the following condition have to stand for the perturbed system to be valid

$$E(\Omega_0) = Fv_{air} \tag{5.22}$$

Taking into consideration the dissipated energy, the Equation 5.21 is more accurately represented as

$$J\ddot{\Omega}\Omega + P_E(\Omega) + b_d|\Omega|\Omega^2 + F\dot{x} = 0 \tag{5.23}$$

where b_d is the coefficient of power dissipation and P_E is the rotor shaft power. Consider the perturbation around equilibrium

$$\dot{x}(t) = \dot{x}_0 + \delta\dot{x}(t) \tag{5.24}$$

$$\Omega_{ref}(t) = \Omega_0 + \delta\Omega_{ref}(t) \tag{5.25}$$

Equation 5.23 can be linearized around equilibrium operating point, $(\Omega_0, \dot{x}_0)$ and represented as

$$J\Omega_0\delta\dot{\Omega} + P'_E(\Omega_0)\delta\Omega + 3b_d|\Omega_0|\Omega_0\delta\Omega + F(\Omega_0, \dot{x}_0)\delta\dot{x} + \dot{x}_0\delta F = 0 \tag{5.26}$$

Let

$$\dot{x}_0\delta F = [K - F(\Omega_0, \dot{x}_0)]\delta\dot{x} \tag{5.27}$$

and Equation 5.26 can be written in s-domain

$$\begin{aligned}\delta\Omega_{ref}(s) &= -\frac{K}{J\Omega_0 s + P'_E(\Omega_0) + 3b_d|\Omega_0|\Omega_0}\delta\dot{x}(s) \\ &= -\frac{K_e}{D(s)}\delta\dot{x}(s)\end{aligned} \tag{5.28}$$

where

$$D(s) = Ts + 1 \tag{5.29}$$

$$T = \frac{J\Omega_0}{P'_E(\Omega_0) + 3b_d|\Omega_0|\Omega_0} \tag{5.30}$$

$$K_e = \frac{K}{P'_E(\Omega_0) + 3b_d|\Omega_0|\Omega_0} \tag{5.31}$$

Equation 5.28 can be seen as a feeding a low passed $\delta\dot{x}(s)$ signal to augment the rotor speed increment consistent with the following block diagram

In which the error dynamics has become

$$\Omega_e = \Omega_{ref} - \Omega_r \tag{5.32}$$

To implement, the perturbed variables are discretized as

$$\delta\dot{x}_k = \dot{x}_k - \dot{x}_0 \tag{5.33}$$

$$\delta\Omega_k = \Omega_k - \Omega_0 \tag{5.34}$$

Equation 5.28 discretized using Euler forward as

$$\delta\Omega_{k+1} = -\frac{\Delta t K_e}{T}\delta\dot{x}_k - \left(\frac{\Delta t - T}{T}\right)\delta\Omega_k \tag{5.35}$$

and finally the augmented reference rotor speed is fed into the blade pitch feedback controller in baseline as

$$\Omega_{k+1} = \Omega_k + \delta\Omega_{k+1} \tag{5.36}$$

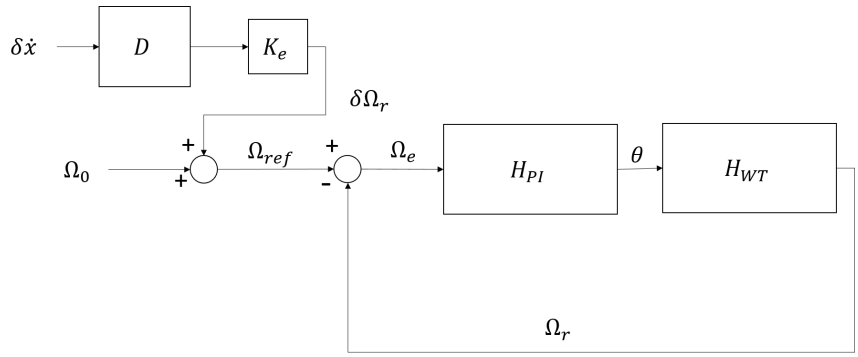

Figure 5.7: Simplified block diagram of energy shaping controller

5.4.2 Controller Tuning

For constant torque (Q_0) application in region 3,

$$P'_E(\Omega_0) = Q_0 = \frac{P_0}{\Omega_0} \tag{5.37}$$

The time constant T can be expressed as

$$T = \frac{J\Omega_0}{\frac{P_0}{\Omega_0} + 3b_d|\Omega_0|\Omega_0} = \frac{J\Omega_0^2}{P_0 + 3b_d|\Omega_0|\Omega_0^2} \tag{5.38}$$

$$K_e = \frac{K}{P'_E(\Omega_0) + 3b_d|\Omega_0|\Omega_0} = \frac{K\Omega_0}{P_0 + 3b_d|\Omega_0|\Omega_0^2} \tag{5.39}$$

such that the values can be taken as $T = 0.048$, $K_e = 62$.

5.5 Individual Pitch Controller

The individual pitch controller (IPC) is designed with the objective to reduce the bending loads on rotor blades through reducing rotor plane yaw and pitch moments and therefore increase their fatigue lives. The blade-pitch actuator controller is intended to be applied in conjunction the variable pitch controller and the generator torque controller to further manipulate the blade pitch individually to reduce the loads on the blades. As described in [3], a rotation matrix is first used to transform flapwise bending moment of the blades into non-rotating coordinates before being fed into a PID controller as input. The outputs of the PID controller are individual blade pitch angle aim to reduce flapwise bending. The Coleman transformation [38] can be expressed as

$$\begin{bmatrix} M_{y1}^{cm} \\ M_{y2}^{cm} \\ M_{y3}^{cm} \end{bmatrix} = \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 \\ 2sin\psi_1(t) & 2sin\psi_2(t) & 2sin\psi_3(t) \\ 2cos\psi_1(t) & 2cos\psi_2(t) & 2cos\psi_3(t) \end{pmatrix} \begin{bmatrix} M_{y1} \\ M_{y2} \\ M_{y3} \end{bmatrix} \tag{5.40}$$

where

- M_{y1}, M_{y2} and M_{y3} are the flapwise bending moments of each blade in rotating frame
- M_{y1}^{cm} is the average blade root flapwise bending moment
- M_{y2}^{cm} is the yaw moment exerted by the blades on the fixed hub of the rotor
- M_{y3}^{cm} is the tilt moment exerted by the blades on the fixed hub of the rotor
- $\psi_i(t)$ are the azimuthal angles of individual blades in a 3-bladed turbine configuration. $\psi_i(t) = 0$ when the blades are positioned vertically upwards

M_{y2}^{cm} and M_{y3}^{cm} are then being used as inputs for a controller. One of the simplest implementation would be to use PID controllers which take the following form and easily be visualized using Figure 5.8

$$\theta_i^{cm} = K_P M_{yi}^{cm} + K_I \int_0^t M_{yi}^{cm} dt + K_D \frac{dM_{yi}^{cm}}{dt} \tag{5.41}$$

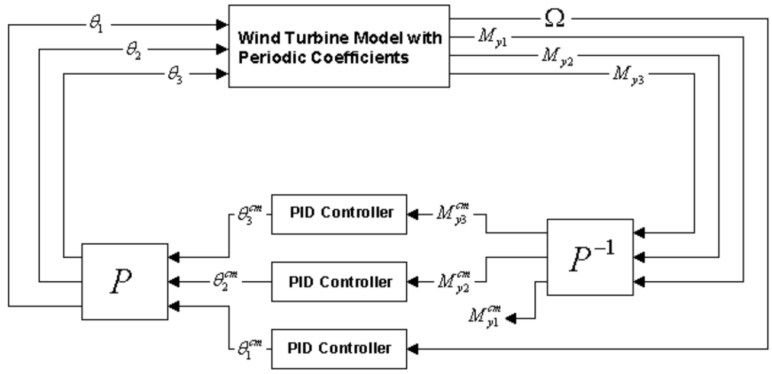

Figure 5.8: Simplified block diagram of an individual pitch controller [3]

The control outputs are then being transformed to rotating frame using the inversed Coleman matrix, as in

$$\begin{bmatrix} \theta_1(t) \\ \theta_2(t) \\ \theta_3(t) \end{bmatrix} = \begin{pmatrix} 1 & 2sin\psi_1(t) & 2cos\psi_1(t) \\ 1 & 2sin\psi_2(t) & 2cos\psi_2(t) \\ 1 & 2sin\psi_3(t) & 2cos\psi_3(t) \end{pmatrix} \begin{bmatrix} \theta_1^{cm}(t) \\ \theta_2^{cm}(t) \\ \theta_3^{cm}(t) \end{bmatrix} \tag{5.42}$$

where

- θ_1, θ_2 and θ_3 are the individual pitch angle signal for each blade in rotating frame
- θ_1^{cm} is the collective (average) blade pitch angle
- θ_2^{cm} is the differential pitch angle in the yaw-wise axis orthogonal to θ_3^{cm}

- θ_3^{cm} is the differential pitch angle in the tilt-wise axis orthogonal to θ_2^{cm}

Similar to the baseline controller, the blade position limits of IPC is between 0° to 90° and the pitch rate is set to $8^\circ/s$. And like the baseline's collective pitch controller, it is only implemented in above rated wind condition (ie region 3).As soon as the individual pitch angles are calculated from the controller which will then be superimposed on the collective pitch angle as in the case of a baseline controller [3]. Due to a bigger blade pitch fluctuation under IPC, threshold on the minimum blade pitch region 3 torque calculation has been increased from 1 to 3° to avoid racing of speed regulation control and IPC at low blade pitch values.

5.5.1 Controller Tuning

The gains are calculated by Lackner et al using iterative method [3]. The values of the proportional, integral, and derivative gains are shown in Table 5.1.

K_P	K_I	K_D
0.06	0.01	-0.001

Table 5.1: PID gain values for IPC

These gains are to be scheduled according to [3]

$$G_{IPC} = \frac{1}{1 + 4.55\theta_c} \tag{5.43}$$

where θ_c is the collective pitch angle in radians.

Chapter 6

Results and Discussions

To evaluate the performance of the controllers, step wind tests with no wave and turbulent wind irregular wave simulations have been conducted in accordance to methods outlined in Chapter 3. This section aims to display the results and discuss in detail the behavior of each controller from different perspectives. First, results step wind simulations are presented, followed by the results for global analysis and MBD analysis of different load cases and finally the comparison of all performance parameters is presented.

6.1 Step Wind

One major benefit of constant wind simulations is that the time required is much shorter and they provide quick evaluations of how the controllers are reacting to step inputs. Through removing wave frequency components, the transient behaviors in terms of signal overshot and the time taken for the transient movement to die out can be clearly captured. Since the controller is designed to act above rated wind speed, wind steps of $2m/s$, $4m/s$, $6m/s$ and $8m/s$ were given to a wind turbine initialized and operating at a wind speed of $12m/s$. A few time series responses of the controller used to describe the main characteristic of the controller are as shown in Figure 6.1. It consists of the plot of wind velocity at the hub, rotor speed and surge motion.

As observed from the rotor speed time history, the rotor speed stabilizes at $1.26rad/s$ for all controller designs with the ES controllers (with and without IPC) having the highest speed excursion at each step. The phenomenon can be explained through the corresponding blade pitch plot. As wind speed increases, it is observed that the blade pitch transition is more gradual for the ES controllers as compared to the baseline and AD controllers. This slower reaction of blade pitch (resulted from the control objective to reduce surge and pitch motions) allows a higher speed excursion as it varies the speed reference at every time step based on the nacelle surge velocity.

At $2400s$, a clear distinction between the ES controller with IPC and the ES controller without IPC can be observed. A speed step of $6m/s$ proves to have exceeded the limit of the controller causing the rotor to stall eventually and the blade pitch fully feathered.

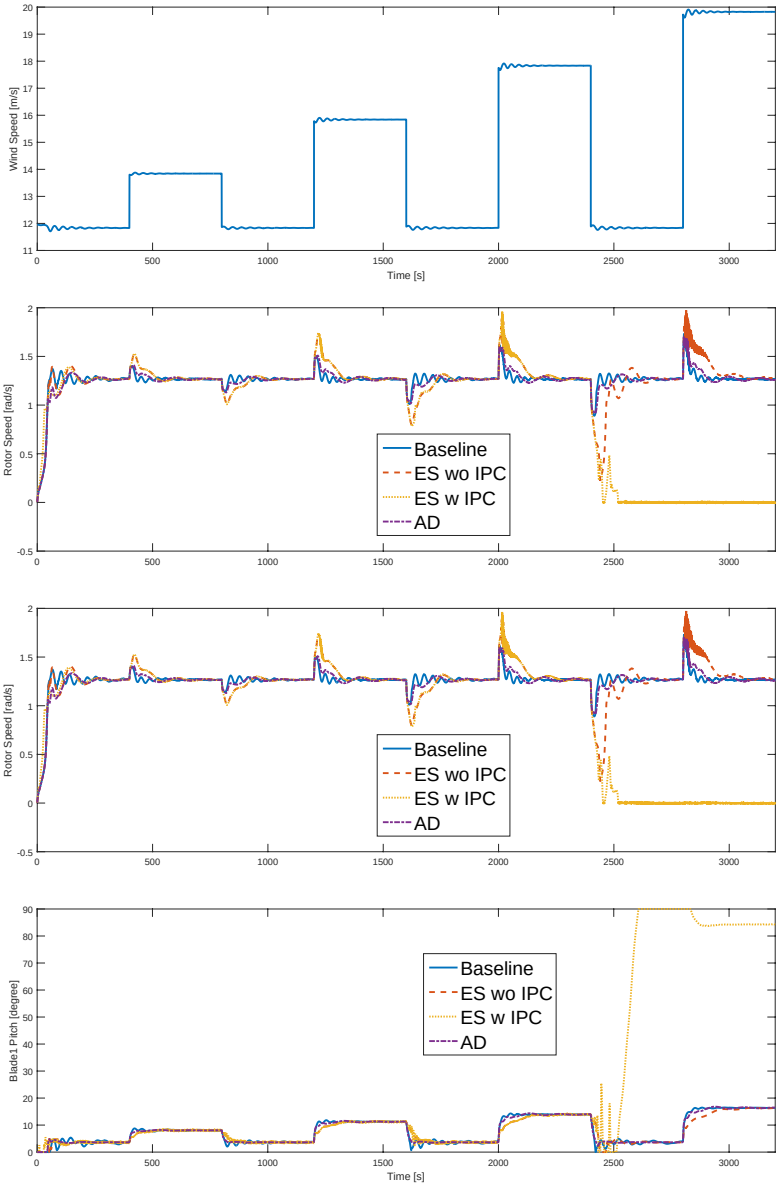

Figure 6.1: Step wind time series from $12m/s$ to $20m/s$

The phenomenon is due to the contradicting control objectives of speed regulation and blade root moment reduction. For example, as the rotor is experiencing a lower speed than the control reference, the speed regulation mechanism works to reduce the blade pitch in order to provide additional torque. However, as the blade pitch is reduced, a higher blade root bending moment is experienced and since the IPC is working in conjunction with

the ES controller, it will call for an increment in blade pitch setting. At one point, the contradicting effects cancel each other out causing an unstable control situation. In later sections, similar effect can be observed in the simulations for global analysis.

6.2 Seed Comparison

Since five unique wave-wind seed combinations were used for the simulation of each environmental conditions, their spectral diagrams for blade-1 flap-wise-root bending moment are plotted to ensure that the responses of any seed combination are accurate representations of the corresponding environmental conditions.

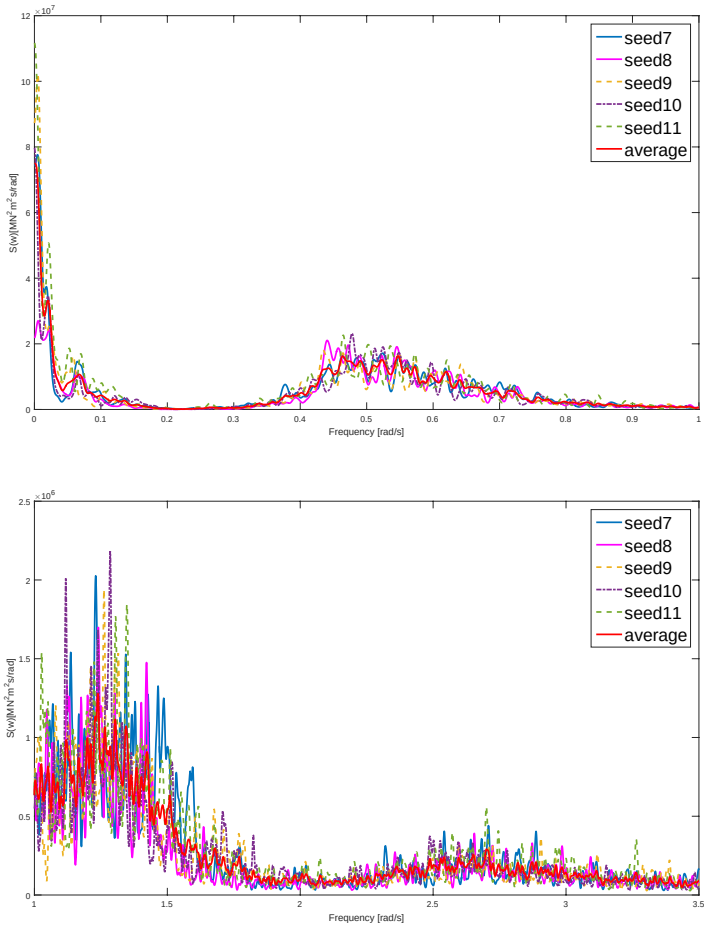

Figure 6.2: Spectral diagram of blade 1 flap-wise-root bending moment for ES controller with IPC in EC4

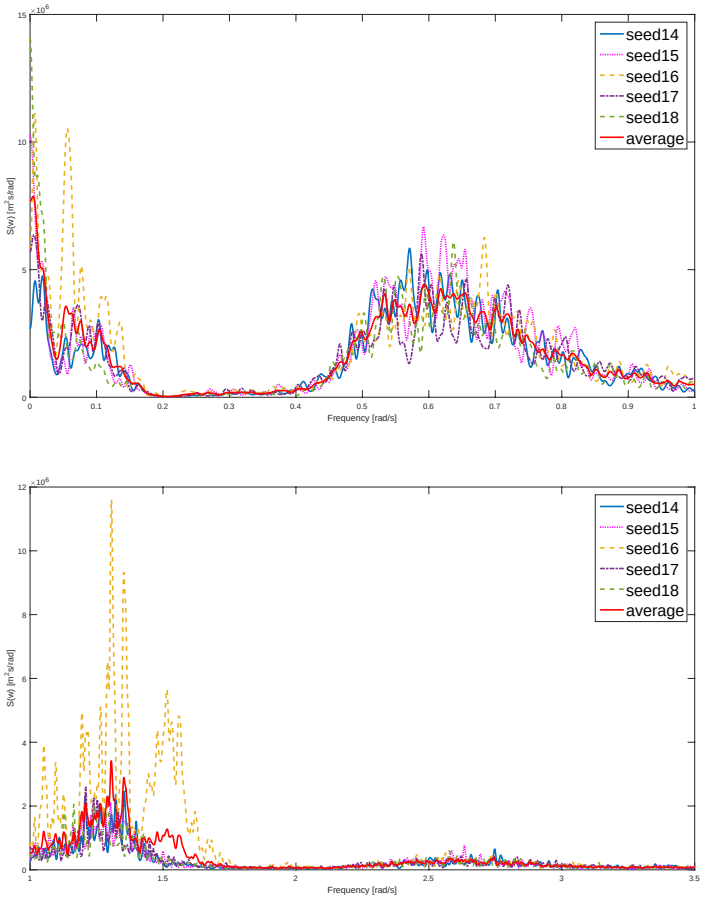

Figure 6.3: Spectral diagram of blade 1 flap-wise-root bending moment for ES controller with IPC in EC5

As shown in Figures 6.2, 6.3 and 6.4, an average spectral curve of the five seeds used is plotted at each spectrum diagram. It is observed that at very low frequency region (below $0.1rad/s$), deviations from the average values are higher and as frequency increases difference between the seeds decreases showing similar trend of responses. A note should however be made for EC5 in Figure 6.3 as the simulation using seed16 is showing significantly higher than average response at frequencies ranging between $0 - 0.1rad/s$ and $1 - 2rad/s$. A close examination of the time series plots shows that a large wind speed reduction has resulted in a high speed correction which in turn caused momentary spikes in the blade1's flap-wise-root moment.

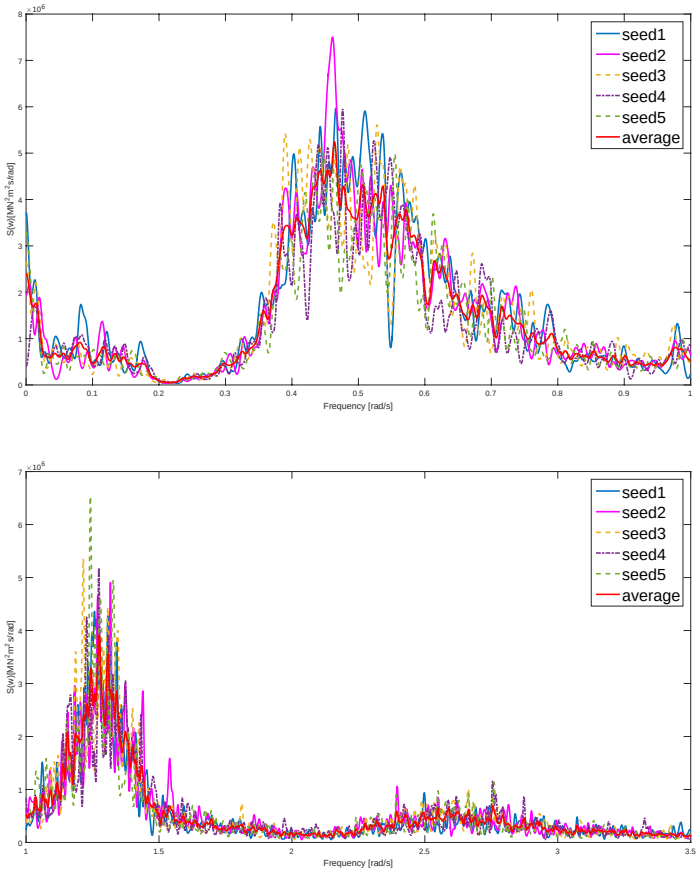

Figure 6.4: Spectral diagram of blade 1 flap-wise-root bending moment for ES controller with IPC in EC6

6.3 Global Analysis

Global responses for three environmental load conditions are summarized in this section. Since all the seeds exhibit similar behaviors, one seed per environmental condition is chosen to represent each group of simulations. The seed-combinations used for each environmental conditions are summarized in Table 6.1.

ECs	Wave Seed	Wind Seed 1	Wind Seed 2
EC4	8	-1337032771	-503679729
EC5	17	526026488	373854159
EC6	4	-1279905479	1264423003

Table 6.1: Wind and wave seeds combinations

6.3.1 Time Series Representation

In order to better illustrate the frequency response using different controllers, a fraction of 1-hr simulation time series in EC6 is first chosen to give an idea the difference in behavior each controller has as compared to one another. In Figure 6.5, the blade pitch, tower base fore-aft bending moment, blade 1 flap-wise bending moment and platform surge motion under the influence of all controller is presented. As shown in the time series, the blade pitch fluctuation using IPC is significantly higher than the rest of the controllers. It managed to also reduce blade root flap-wise moment significantly with an increase in response for tower fore-aft bending moment. Surge motion is reduced using all versions of controllers.

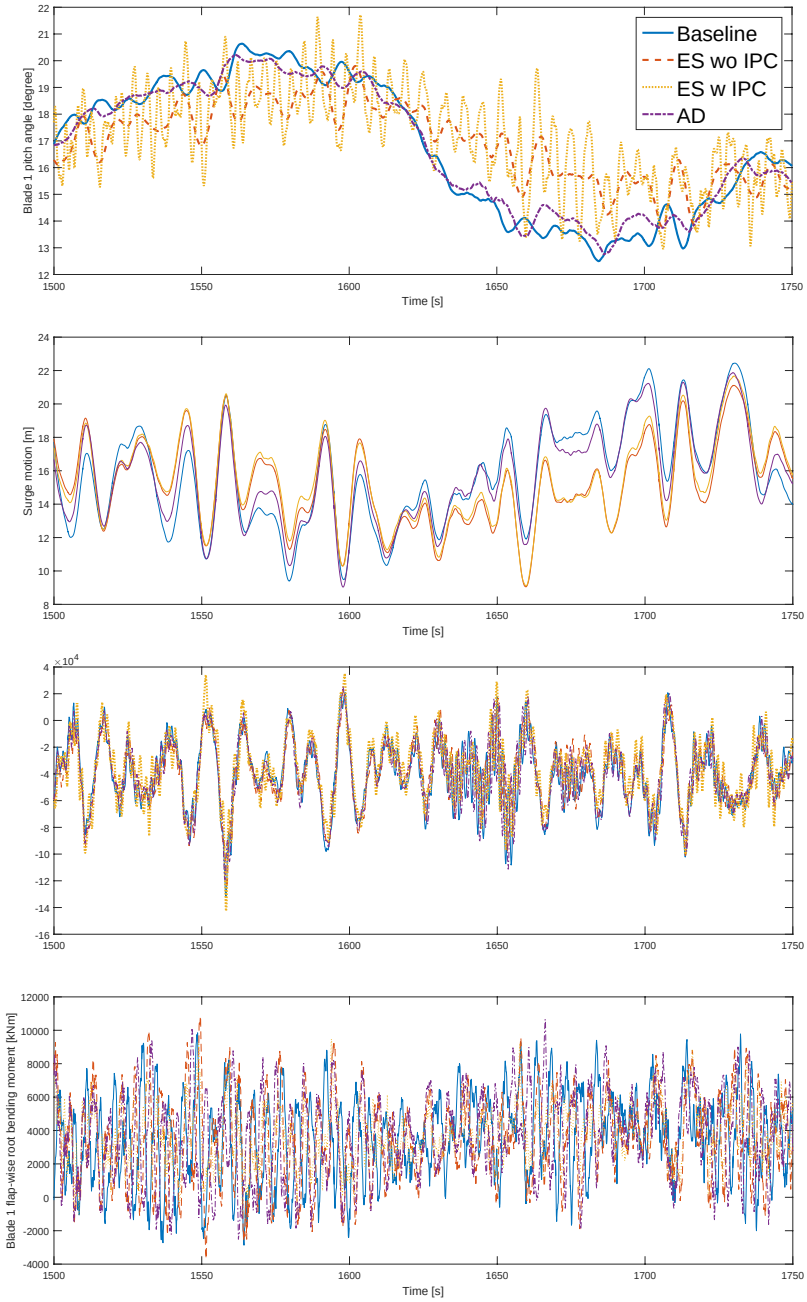

Figure 6.5: Time series comparison of different controllers under EC6

6.3.2 Surge and Pitch Motions

One major modification to the baseline controller aims at reducing the surge and pitch motion by introducing damping through the nacelle surge velocity feedback. It is therefore important to compare the effects each controller has on the platform motions. The surge spectral diagrams for all environmental conditions are presented in Figures 6.6, 6.7 and 6.8. Note that each spectral diagram is divided into two for higher resolution representation.

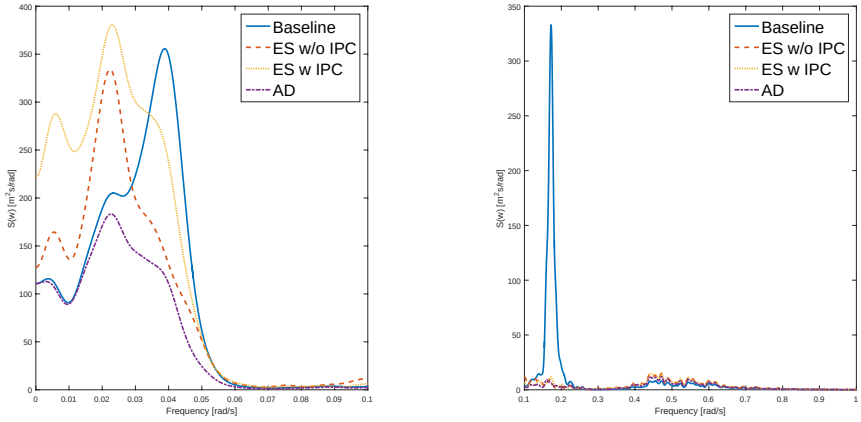

Figure 6.6: Spectral diagram of platform surge motion for all controller in EC4

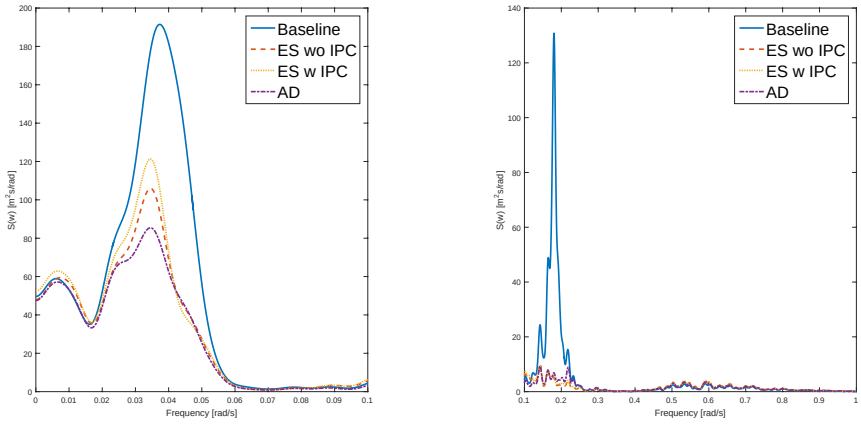

Figure 6.7: Spectral diagram of platform surge motion for all controller in EC5

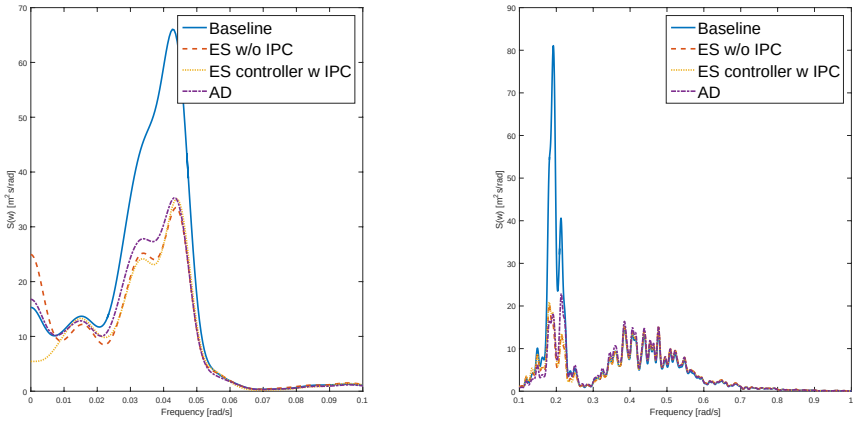

Figure 6.8: Spectral diagram of platform surge motion for all controller in EC6

One major improvement the 3 modified versions of controllers has over the baseline controller is the significant reduction of resonance at frequencies close to the platform's pitch natural frequency ($0.21rad/s$). It is also observed that the surge response due to wind frequencies is reduced for simulations with higher mean wind speed (the left of Figures 6.7 and 6.8). In the left of Figure 6.6, it is shown that the modified controllers still achieved response reduction close to surge natural frequency but when the frequency reduces further, the ES controllers show signs of weakening damping for higher response is observed using the two controllers. It can also be reflected in further frequency response (for example tower base fore-aft bending) that the ES controller is not effective in mitigating response in low (wind) frequency region while the addition of IPC manage to improve that but it is only limited to the responses associated locally with the turbine (for example tower top fore-aft bending).

The platform's pitch motion in all environmental conditions are described by spectral diagrams as shown in Figures 6.9, 6.9 and 6.9.

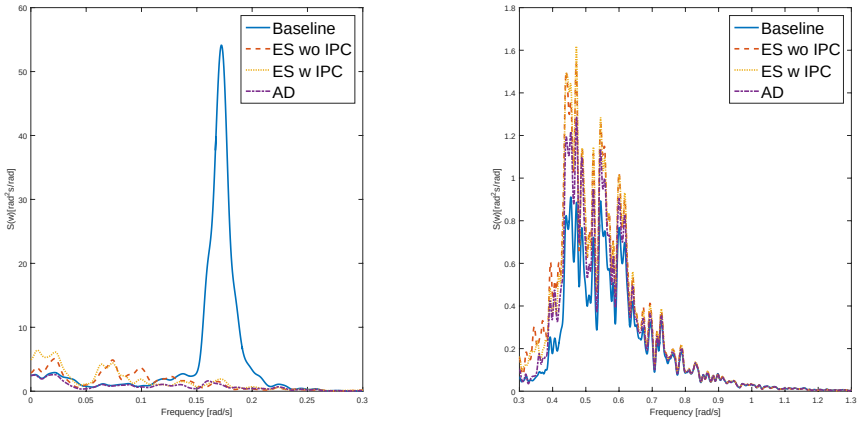

Figure 6.9: Spectral diagram of platform pitch motion for all controller in EC4

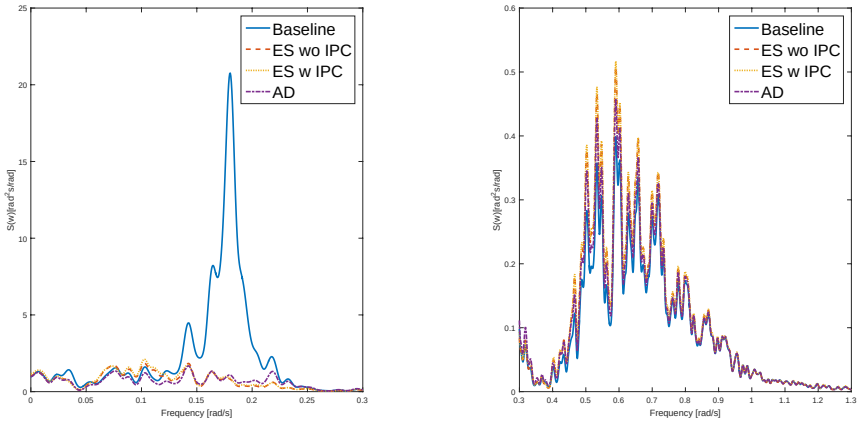

Figure 6.10: Spectral diagram of platform pitch motion for all controller in EC5

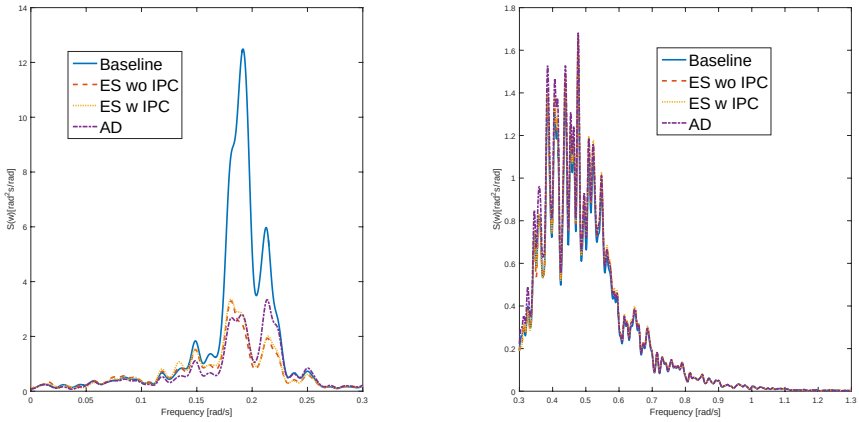

Figure 6.11: Spectral diagram of platform pitch motion for all controller in EC6

As pitch motion is coupled with surge motion, similar trend can be anticipated for the pitch spectral diagrams. The resonance at pitch natural frequency for all 3 environmental load cases has reduced significantly especially for the cases of EC4 and EC5. The pitch behavior at wave frequencies remains largely unchanged as a result of passing nacelle surge velocity through a low pass filter at wave frequency prior to using it as control input.

6.3.3 Tower Base Fore-Aft Bending Moment

The tower base bending moment spectral diagrams are as shown in Figures 6.12, 6.13 and 6.14. The significance of reduced surge and pitch motions is reflected through a drop in fore-aft bending moment response close to pitch natural frequency for all 3 environmental load cases. This reduction of load contributes directly to a reduction in tower fatigue damage making it possible to further upscale turbine sizes. Although fatigue damage at the tower base is distributed non-uniformly around its circumference, due to the environmental conditions defined within this project in which wind and wave were assumed to be unidirectional, the fatigue damage will be concentrated in the fore-aft direction.

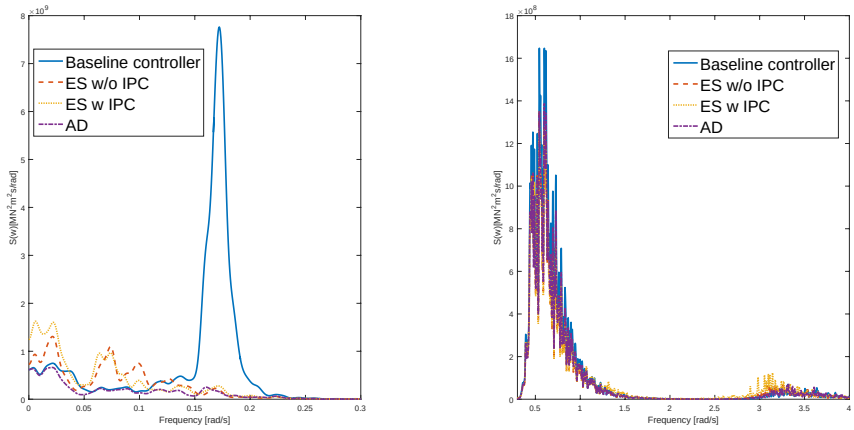

Figure 6.12: Spectral diagram of tower base fore-aft bending moment for all controller in EC4

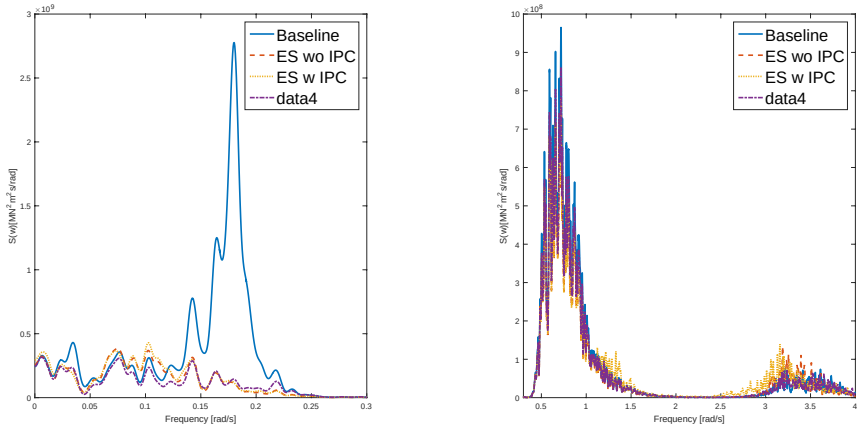

Figure 6.13: Spectral diagram of tower base fore-aft bending moment for all controller in EC5

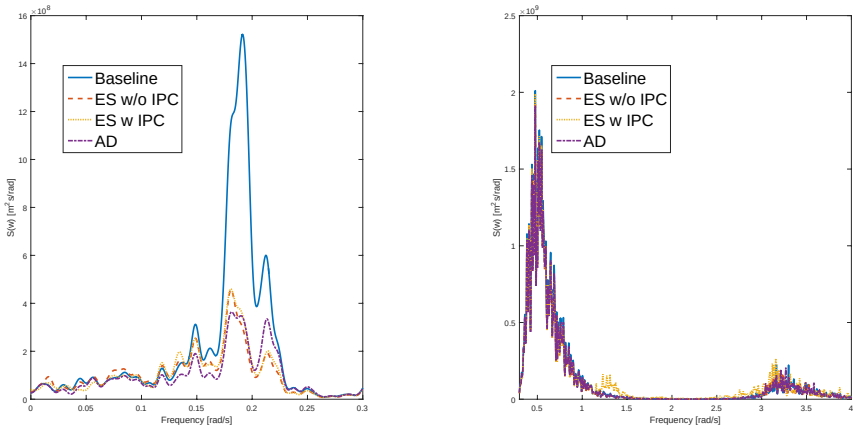

Figure 6.14: Spectral diagram of tower base fore-aft bending moment for all controller in EC6

At the region close to tower fore-aft first mode natural frequency ($3 - 3.5 rad/s$), an increase in response can be observed for the ES controller with IPC but the increments are relatively small as compared to the reduction around pitch natural frequency. A relatively high percentage increase in response for the ES controller with IPC at the frequency close to rotor frequency (1P) is also detected. This is due to the control frequency of the IPC as the measured blade moment is low-passed at 1P prior to being used as input. This is an important characteristic of the IPC as it introduces control interference at 1P which will be encountered in other spectral diagrams below.

6.3.4 Tower Top Fore-Aft Bending Moment

Spectral diagrams of the tower top bending moment in the fore-aft direction are as shown in Figures 6.12, 6.13 and 6.14. It is observed that at frequencies lower than platform pitch natural frequency, IPC has successfully reduced significant amount of resonance. In higher frequency region after IP, slight increase of response is detected for ES controller with IPC but the magnitude is relatively small as compared to the resonance in lower frequencies.

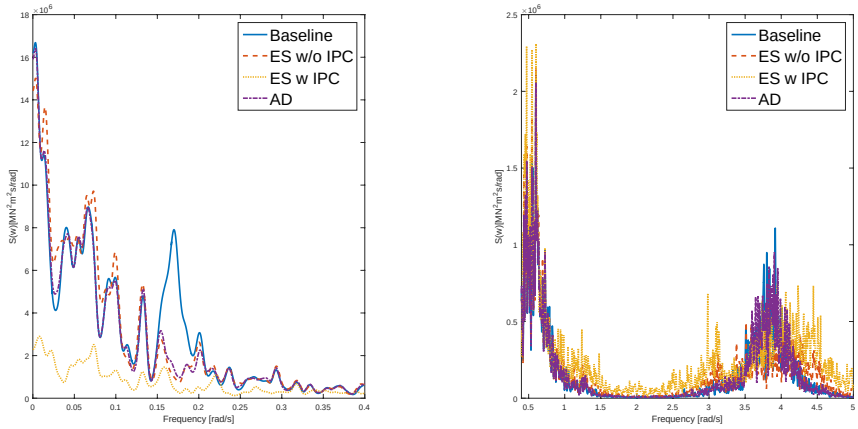

Figure 6.15: Spectral diagram of tower top fore-aft bending moment for all controller in EC4

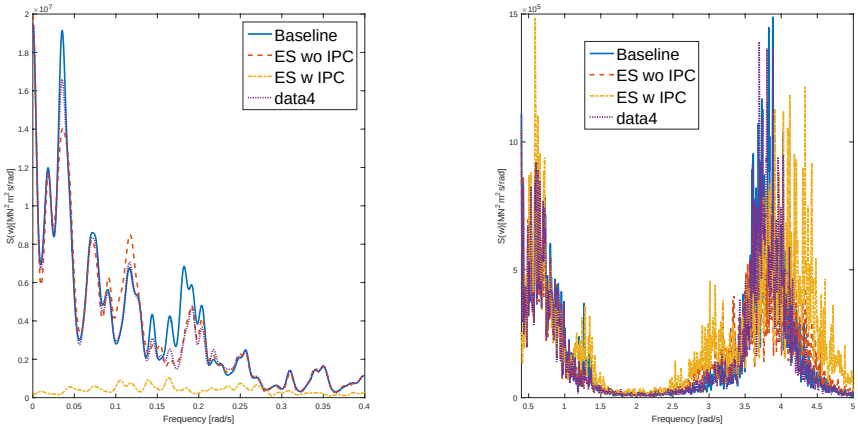

Figure 6.16: Spectral diagram of tower top fore-aft bending moment for all controller in EC5

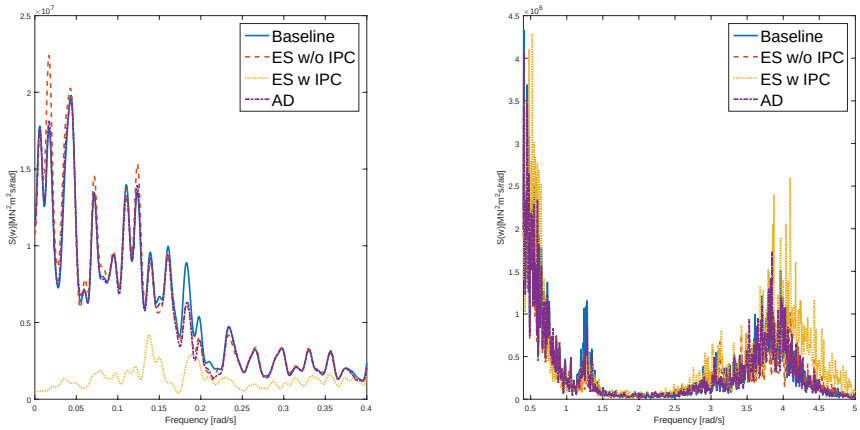

Figure 6.17: Spectral diagram of tower top fore-aft bending moment for all controller in EC6

6.3.5 Blade 1 Bending Moment

Blade moment for one of the three blades are plotted in spectral diagram to evaluate specifically the effect of using an IPC in addition to the ES controller. The spectral diagrams for all environmental load cases are as shown in Figures 6.18, 6.19 and 6.20. In addition to a reduction in the response close to pitch natural frequency as in the case of tower base fore-aft moment, a second reduction at frequencies close to 1P is detected for the ES controller with IPC. An increase in response in the region of wave frequency is however observed for both ES controllers due to increased blade pitch fluctuations close to wave frequency. Comparison can be drawn with the AD controller where there is no added response at wave frequency. A slower blade pitch reaction of the ES controller resulted in added response at wave frequency.

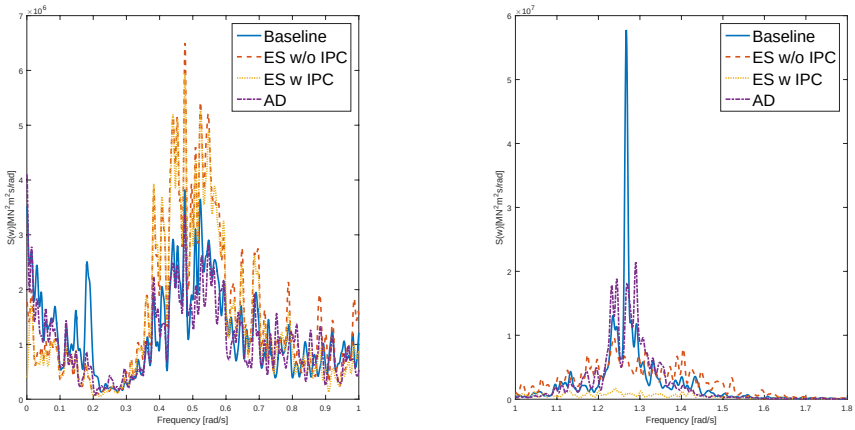

Figure 6.18: Spectral diagram of blade 1 flap-wise-root bending moment for all controller in EC4

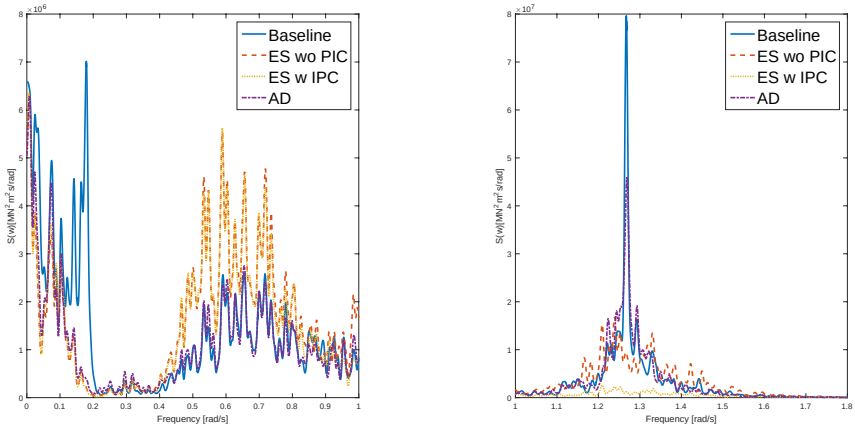

Figure 6.19: Spectral diagram of blade 1 flap-wise-root bending moment for all controller in EC5

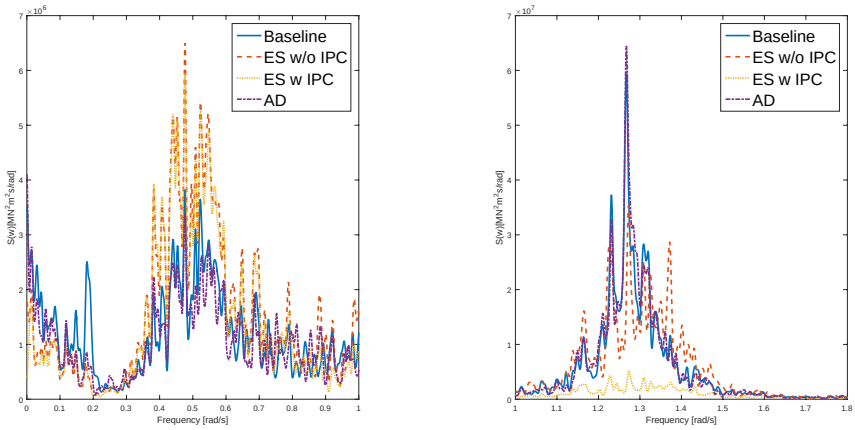

Figure 6.20: Spectral diagram of blade 1 flap-wise-root bending moment for all controller in EC6

6.3.6 Wind Turbine Aerodynamic Pitch and Yaw Moment

The basic working principle of the IPC is to reduce blade root moment through reducing aerodynamic pitch and yaw moment of the rotor plane. It is therefore interesting to look at these two moments to verify the correct implementation of the algorithm. The wind turbine aerodynamic pitch and yaw moment spectral diagrams for all environmental conditions are as shown in Figures 6.21 to 6.26.

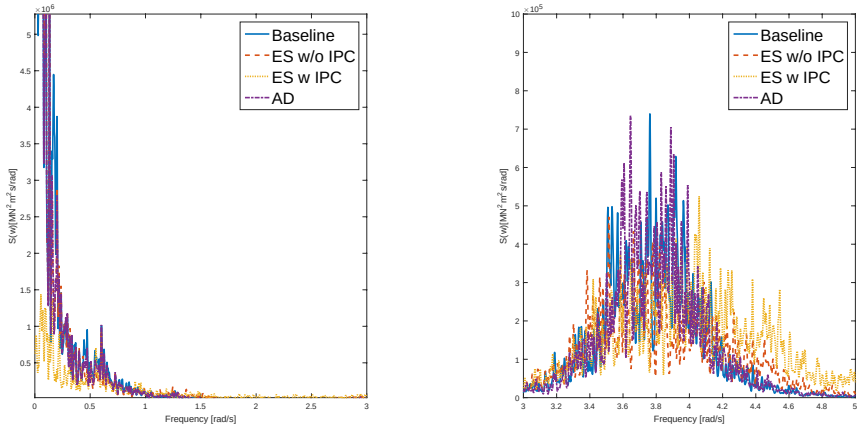

Figure 6.21: Spectral diagram of turbine aerodynamic pitch moment for all controller in EC4

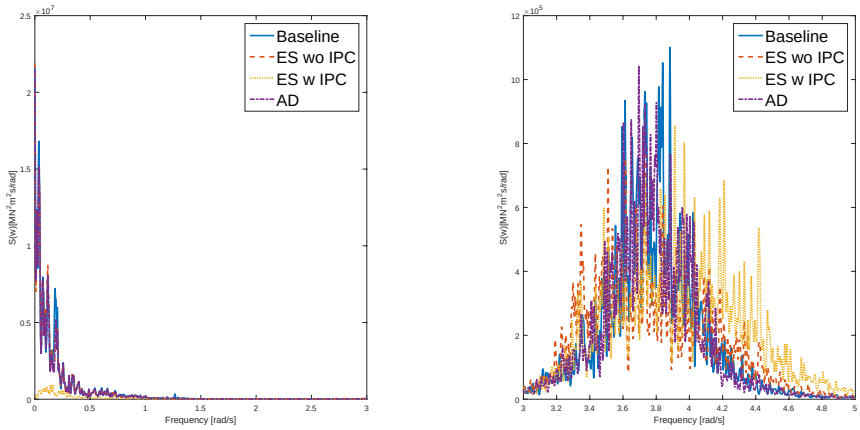

Figure 6.22: Spectral diagram of turbine aerodynamic pitch moment for all controller in EC5

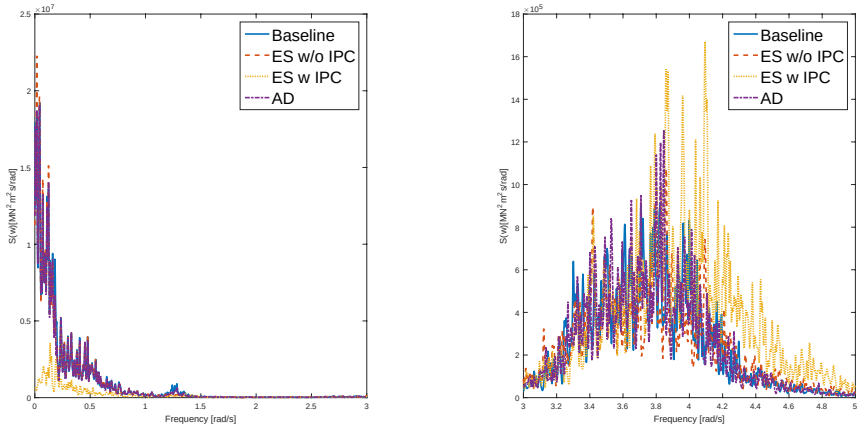

Figure 6.23: Spectral diagram of turbine aerodynamic pitch moment for all controller in EC6

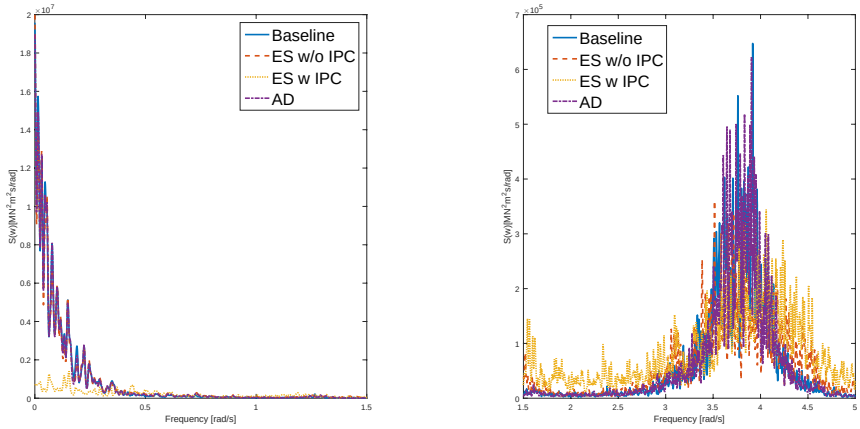

Figure 6.24: Spectral diagram of turbine aerodynamic yaw moment for all controller in EC4

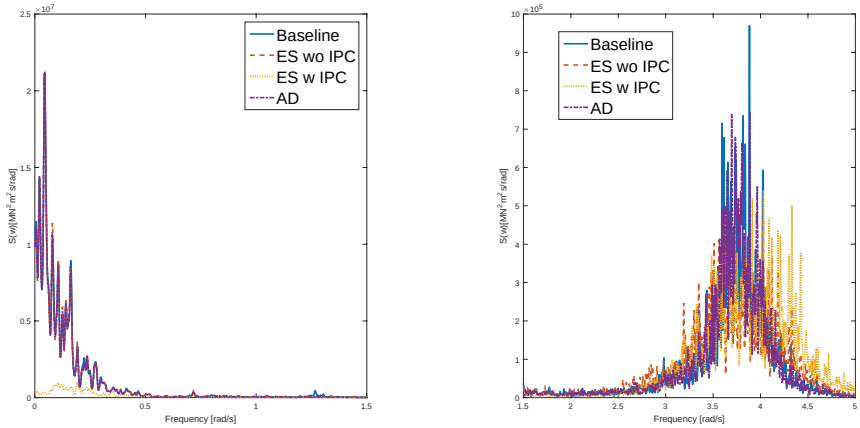

Figure 6.25: Spectral diagram of turbine aerodynamic yaw moment for all controller in EC5

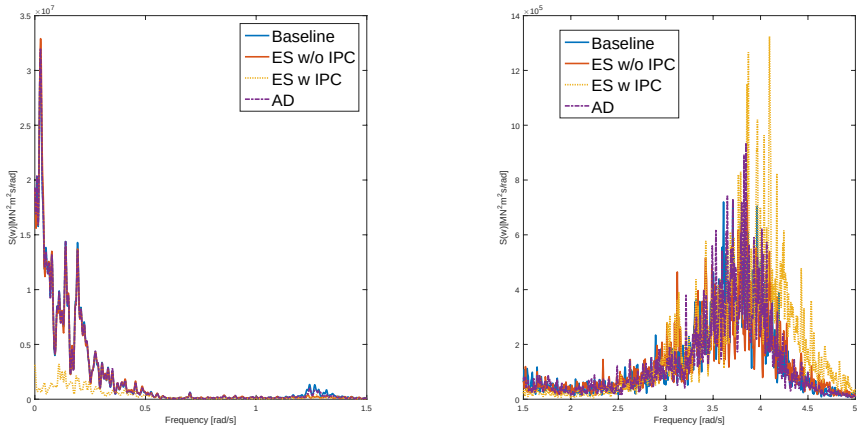

Figure 6.26: Spectral diagram of turbine aerodynamic yaw moment for all controller in EC6

The diagrams show that the IPC is implemented correctly as resonance at frequency region lower than 1P has been reduced significantly. With IPC, there is less tilting and yawing of the rotor plane which contributes directly to a lower tower top fore-aft bending and tower top torsion.

6.3.7 Tower Top Side-Side Shear Force

The tower side-side force, $F_{TowerTop_y}$ is in the main source of contribution for radial load on the bearings and gears. Since the control strategies discussed within this work do not aim at reducing force in this direction, indirect consequences that come with their implementation need to be made aware of. The spectral diagrams of $F_{ToweTopy}$ in all environmental conditions are plotted in Figures 6.27, 6.28 and 6.29.

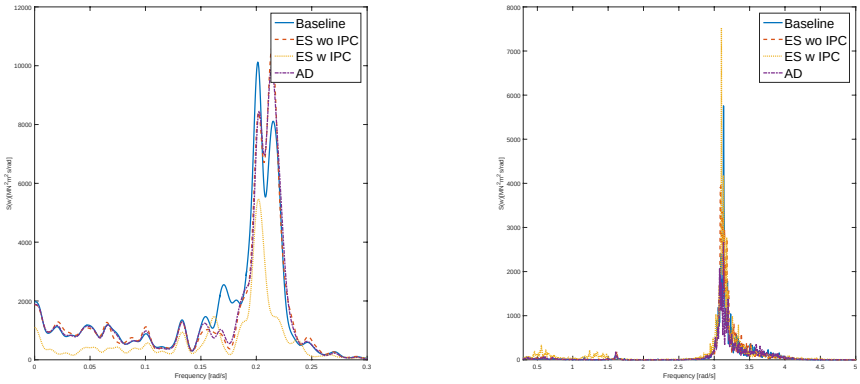

Figure 6.27: Spectral diagram of tower top side-side shear force for all controller in EC4

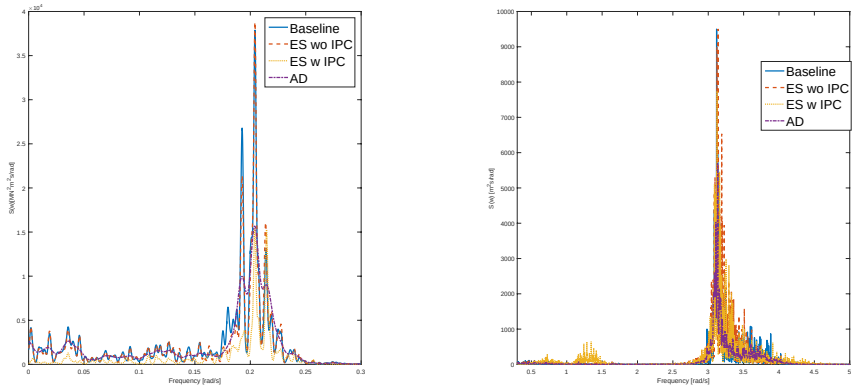

Figure 6.28: Spectral diagram of tower top side-side shear force for all controller in EC5

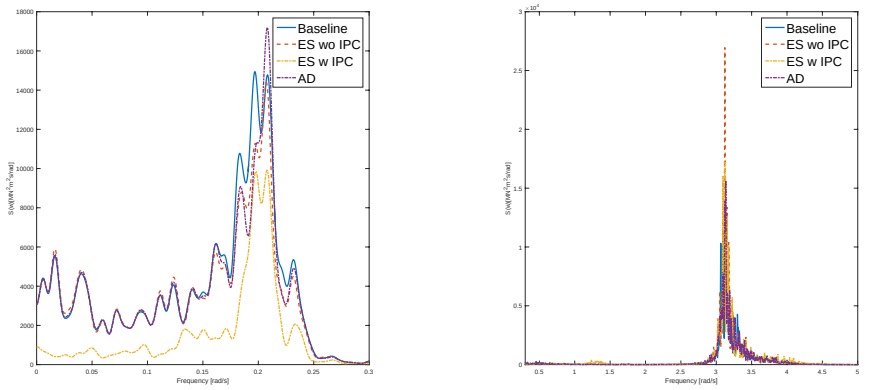

Figure 6.29: Spectral diagram of tower top side-side shear force for all controller in EC6

Significant increment of response concentrated around 1P frequency for ES controller with IPC is observed in all three environmental test cases similar to the trend for tower top fore-aft bending moment. This can be explained as an effort by the controller to reduce aerodynamic moments of the rotor plane at every rotation to keep the rotor plane at its equilibrium position. As the blade pitch is adjusting to change in moment, aerodynamic thrust variation increases which also indirectly resulted in an increase force fluctuation in the side-side direction. The implication of an increase in $F_{TowerTop_y}$ in 1P-frequency is clearly visible in the forces in gear and bearings which will be discussed in the following section.

6.4 Multi-body System Analysis

In order to study the effect of different control strategies on the drivetrain, gear circumferential force and bearing forces time series are extracted from multi-body simulations and plotted in frequency domain.

6.4.1 Gear Circumferential Force

The circumferential force of a gear is a direct contributor of gear fatigue accumulation. In this work, the first stage sun gear of the 5MW gearbox is chosen as an example to study the frequency response of a gear in the drivetrain. The circumferential force for the sun gear under all environmental conditions are as shown in Figures 6.30, 6.31 and 6.32.

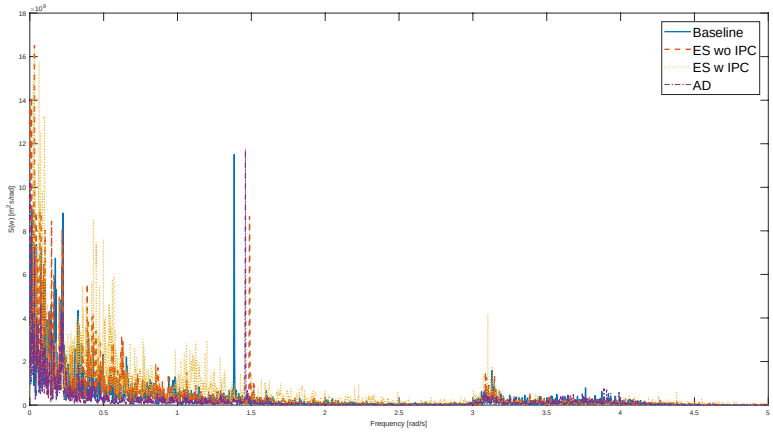

Figure 6.30: Spectral diagram of 1-stage sun gear circumferential force for all controller in EC4

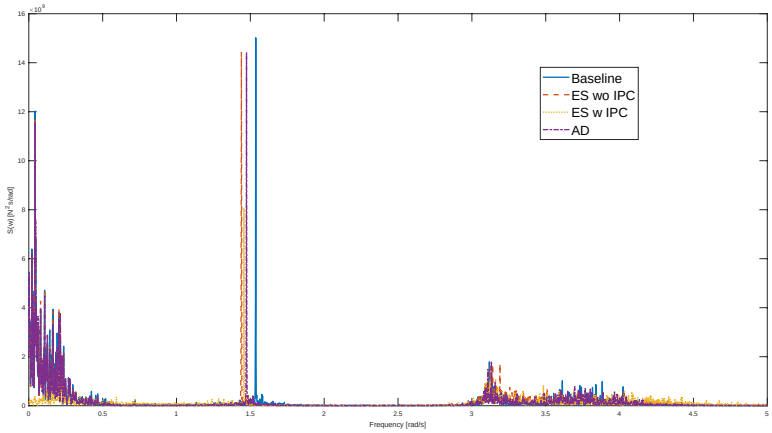

Figure 6.31: Spectral diagram of 1-stage sun gear circumferential force for all controller in EC5

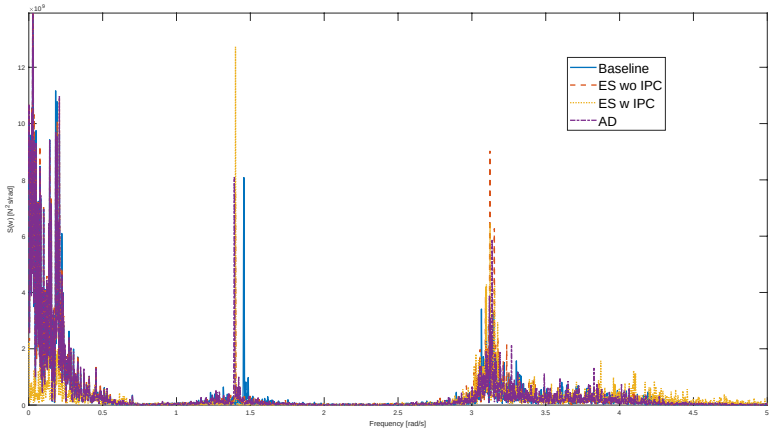

Figure 6.32: Spectral diagram of 1-stage sun gear circumferential force for all controller in EC6

One common feature for all three spectral diagram would be the clear spikes close to $1.5rad/s$ which are caused by gear excitation at 1P. It presents an interesting observation as the speed values deviate from the original input from global analysis by a significant margin. For example, the input speeds from global analysis for EC4 range from 1.252 to $1.267rad/s$ for all controllers while the mean speeds of the main shaft from multi-body system analysis range from 1.38 to $1.48rad/$. This can be due to the effect of an in-built PID-controller used in SIMPACK to regulate speed through torque manipulation. Tower fore-aft first mode natural frequency can also be clearly identified from all three plots. A clear distinction of the spectral diagram for EC4 is the increased response at low frequency

(less than $1.5rad/s$) for the ES controllers except at pitch natural frequency. Since the circumferential force is caused by tower top shear force in y-direction, $F_{TowerTop_y}$ and tower top axial force in z-direction, $F_{TowerRop_x}$ (both in tower local coordinate, refer to Figure 3.7), the two input forces are investigated. Although the spectral diagram for $F_{TowerTop_y}$ 6.27 shows no sign of response at lower frequency for ES controller with IPC, the contribution was found to be from $F_{TowerRop_x}$ in which clear response was detected at lower frequency. On the other hand, low frequency the behavior ES controller with IPC in EC5 and EC6 is more desirable with little response visible (Figures 6.31 and 6.32).

6.4.2 Bearing Axial and Radial Force

The bearing axial and radial forces are directly proportional to its dynamic equivalent load which is used to predict the fatigue life. Frequency response of these forces is therefore important for users to pinpoint the frequencies at which the bearings are reactive to and devise a control strategy accordingly to reduce fatigue damage. The two most important bearings of a gearbox are the main bearings on the main rotor shaft. Placed nearer to the rotor blade, main bearing A (INPA) is designed predominantly to carry radial loads while main bearing B (INPB) which is place further downwind is design to carry most of the axial load. In this work, the loading condition of INPB is chosen to study the effects different controllers have on bearings. The spectral diagram of axial load and radial load in y-direction for INPB in all environmental conditions are as shown in Figure 6.33 through 6.38.

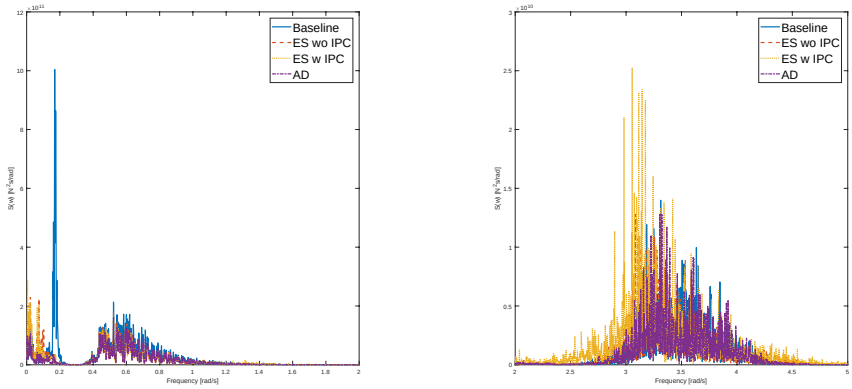

Figure 6.33: Spectral diagram of bearing axial force for all controller in EC4

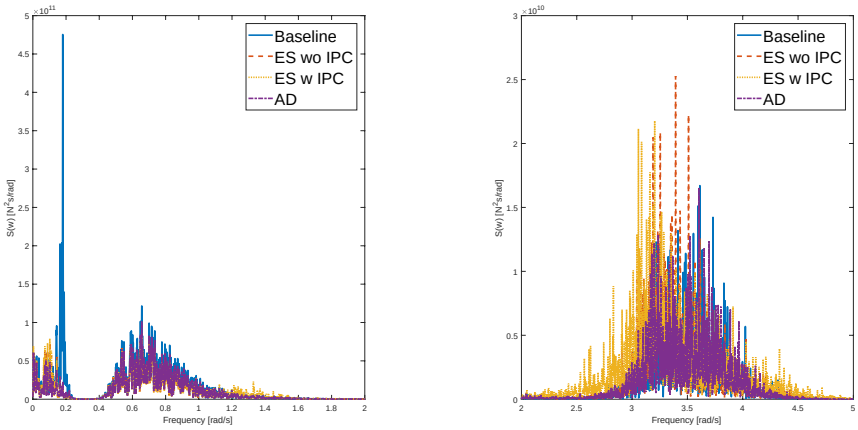

Figure 6.34: Spectral diagram of bearing axial force for all controller in EC5

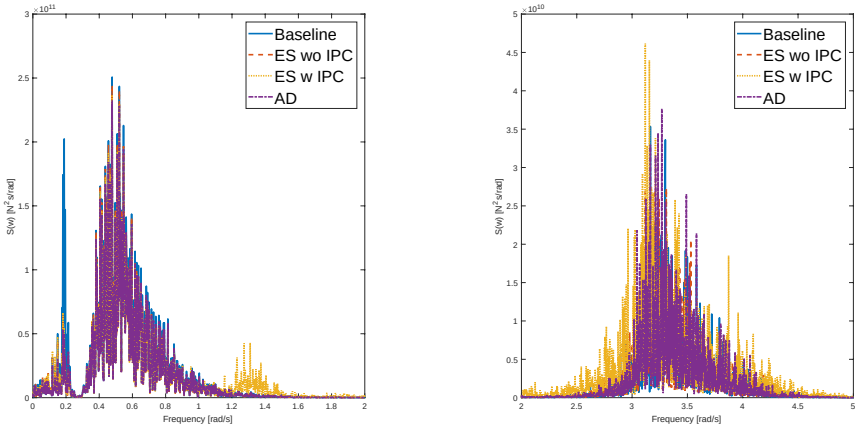

Figure 6.35: Spectral diagram of bearing axial force for all controller in EC6

The frequency response of INPB with all modified versions of controllers is characterized by a significant reduction in response at platform pitch natural frequency in line with motion and structural behaviors in the fore-aft direction (such as surge motion and tower fore-aft bending moment). For ES controller with IPC, increased response in the region of 1P frequency is detected and it is due to an increase in thrust force response in the same frequency region.

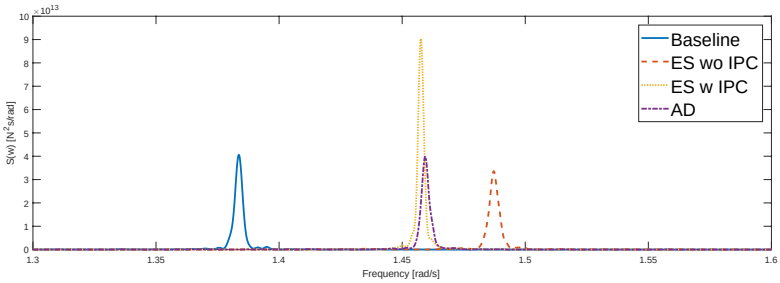

Figure 6.36: Spectral diagram of bearing radial force in y-direction for all controller in EC4

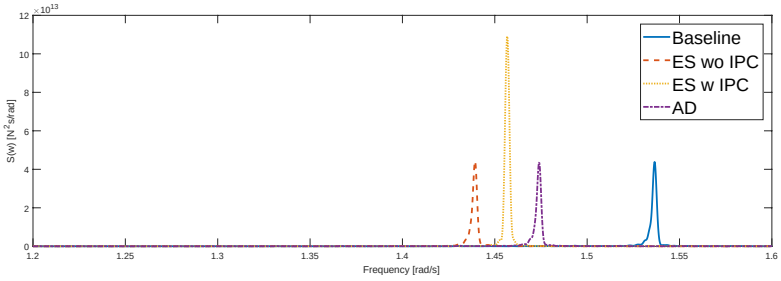

Figure 6.37: Spectral diagram of bearing radial force in y-direction for all controller in EC5

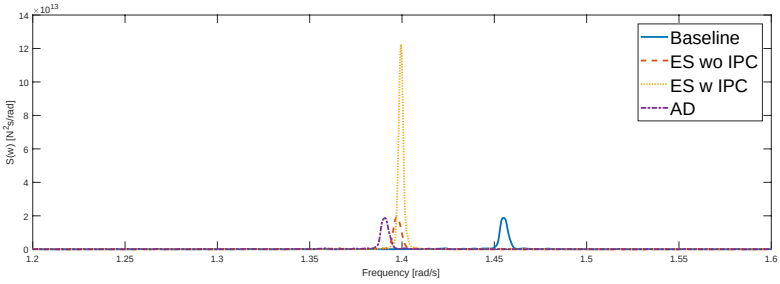

Figure 6.38: Spectral diagram of bearing radial force in y-direction for all controller in EC6

For the radial force in y-direction, sharp peaks are observed at 1P-frequency. As mentioned before, simulation using a different controller will result in a different 1P-frequency. Higher response happens with the use of ES controller with IPC evident of an increased tower side-side shear force response at 1P-frequency.

6.5 Performance Parameters

The performance parameters for performance evaluation of modified controllers are plotted as percentage change as compared to the baseline controller. The tower base 1-hr fatigue damage comparison is as shown in Figure 6.39.

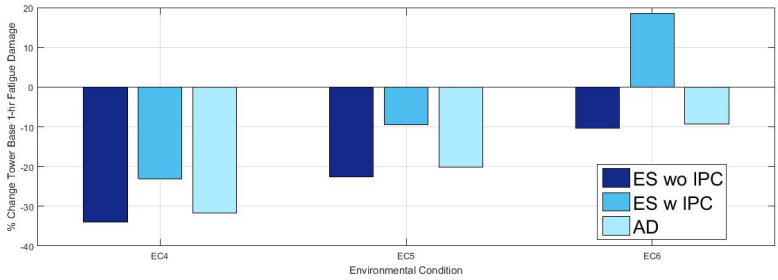

Figure 6.39: Performance comparison - Tower Base 1-hr fatigue damage

As anticipated, all modified controllers managed to reduce the fatigue damage at tower base where ES controller without IPC and AD controller achieving similar reduction in fatigue damage. The introduction of IPC lower the performance of ES controller as it introduces additional response at 1P-frequency. For EC6, the added damage appeared to be too overwhelming as it cancelled off the damage reduction by ES controller. As shown in the spectral diagram of fore-aft tower bending moment for EC6 (Figure 6.14), EC6 having the highest amount of response among all environmental conditions further strengthen the point.

In terms of blade pitch actuator duty cycle, the performance of the controllers in all environmental conditions is mixed. However, ES controller with IPC is expected to perform poorly due to the added blade pitch activity to regulate the rotor plane moments but in EC5 it manage to reduce actuator activities by more than 5%. A more than 50% reduction in pitch activity for the ES controller without IPC in EC4 comes as a surprise as one might expect that with additional control objective, there would be an increase in actuator activity.

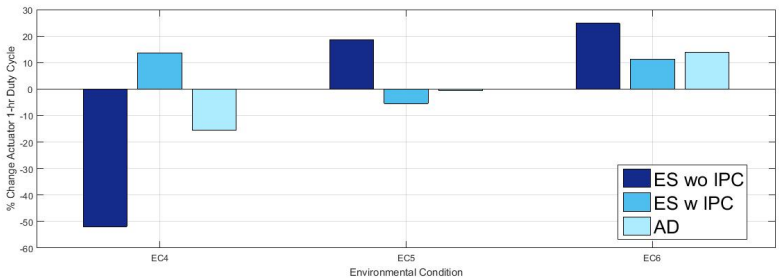

Figure 6.40: Performance comparison - Blade pitch actuator duty cycle

As shown in Figure 6.41, ES controller with IPC is the only controller with significant fatigue damage reduction with EC6 recorded a reduction more than 40%. Performance of the controller in EC4 has been limited by an increase gear circumferential force response in the low frequency range and EC5 by the increase tower top side-side shear force response at 1P-frequency.

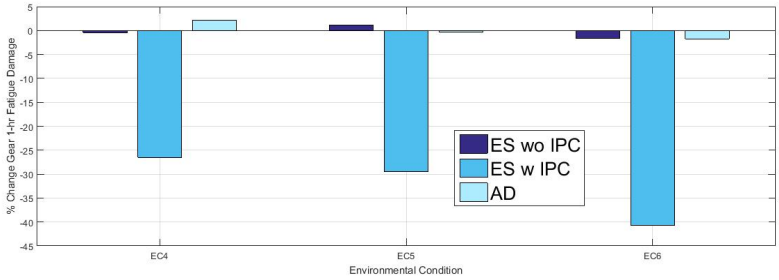

Figure 6.41: Performance comparison - First stage sun gear 1-hr fatigue damage

The performance comparison for main bearing 1-hr fatigue damage is as shown in Figure 6.42. Decrease performance as wind speed increases is due to the increase response of radial and axial bearing forces at 1P-frequency.

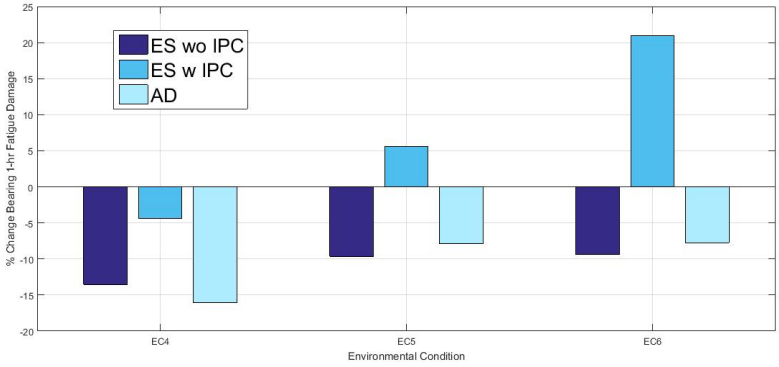

Figure 6.42: Performance comparison - Main bearing (INPB) 1-hr fatigue damage

The performance comparison for blade root 1-hr fatigue damage is as shown in Figure 6.43.

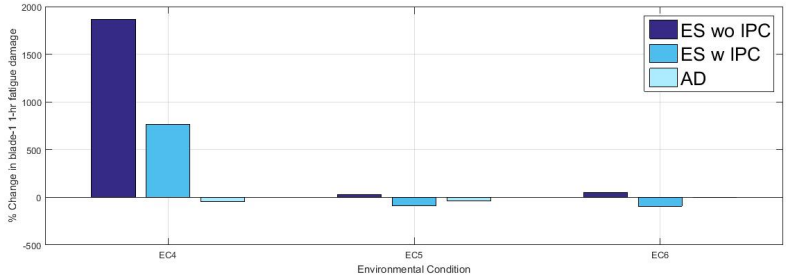

Figure 6.43: Performance comparison - Blade root 1-hr fatigue damage

The calculation for blade root bending clearly yield unrealistic results with more than 1800% increase in fatigue damage using ES controller with IPC for EC4. One possible reason can be due to a high sensitivity M-N curve slope being used. Nevertheless, the trend can still be justified through other characteristics such as time series plots, standard deviation and frequency response. The ES controller increases the blade root moment response at 1P-frequency and hence the increased damage using the controller. However, when IPC is being turned on, the damage is greatly reduced in agreement with the smaller moment amplitude throughout the time series plots. AD controller generally reduces the fatigue in blade.

The mean and standard deviation of power output are as shown in Figure 6.44 and Figure 6.45.

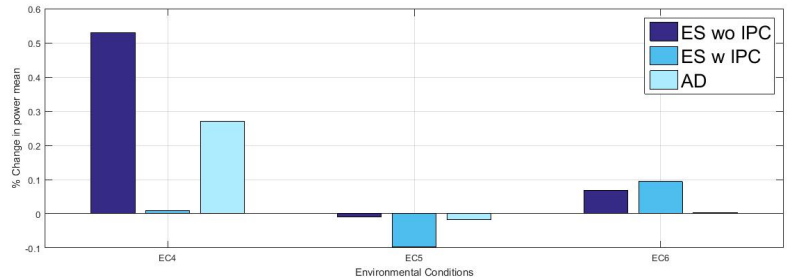

Figure 6.44: Performance comparison - Mean of power

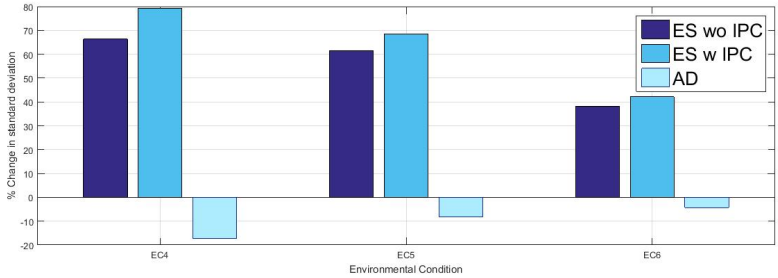

Figure 6.45: Performance comparison - Standard deviation of power

The power mean using all three different controllers does not vary much as compared to the baseline controller with the highest change recorded being less than 0.6%. The standard deviation on the other hand varies significantly especially for the ES controllers. This is in line with the controller characteristic as being discuss in Chapter 6 that the ES controller allows higher rotor speed excursion and hence the higher power deviation. ES controller is observed to favour high wind speed as the standard deviation becomes smaller (can be linked with ES not doing anything for low wind speed in conclusion, look at surge spectral). AD controller on the other hand allows lower speed excursion and thus a lower power fluctuation.

Chapter 7

Conclusion and Recommendations for Future Work

7.1 Conclusion

This thesis started with the intention to implement in the aero-hydro-servo-elastic code several mathematically proven alternative control algorithms aimed to reduce control-induced resonance of floating offshore wind turbines. Despite modifications made on conventional land-based wind turbine controller to adapt to floating environment, it was proven that improvement can still be made with the used of existing or new measurement data.

One common advantage that comes along implementing all the alternative controller designs is an improved motion response in surge and pitch directions which results in obvious load resonance reduction at the platform's pitch natural frequency. Results show that the Active Damping (AD) controller and Energy Shaping (ES) controller achieved similar reduction of resonance. However, the mathematical model of the ES controller causes it to be dependent of rotor speed and thus able to capture frequencies that are below rotor frequency regardless of the frequency range of the input. Blade root flap-wise bending moment is one of the examples of load excitation at frequency lower than rotor frequency (specifically at wave frequency even though the input contains platform wind frequency) using ES controller as compared to the result using AD controller. The implementation of individual pitch controller (IPC) successfully reduce blade root flap-wise bending moment resonance at rotor frequency but at the same time introduces excitation of tower top shear force at rotor frequency. The reduced blade root moment therefore comes with a cost of increased radial load resonance in drivetrain gears and bearings.

Effort made to evaluate performance on the drivetrain through MBS analysis proves to be fruitful as it opens up new areas of concerns to be considered while performing global analysis. Tower top shear stress for instant that may not have been perceived as an impor-

tant design criteria from a structural load perspective, was found to be contributing to radial load of bearings and gears and should be taken into consideration in controller design. Interestingly, fatigue performance evaluation shows some discrepancy with initial estimation of controller performance deduced from frequency response. For tower base fore-aft bending on ES controller with IPC turned on, the increase resonance at 1P-frequency proves to be damaging enough to offset huge amount of the load reduction effect due to reduced surge and pitch motions. This eventually contributes to an increased fatigue damage on the main bearing using the same controller.

To conclude, this paper further validates the workability of some state of the art controller designs. Rather than coming to a conclusion that one particular controller stands out, it is the intention of this paper to draw attention to exploring more criteria to form a basis for comparison. With all possible aspects considered, optimization cost functions can then be utilized to select one controller that performs best.

7.2 Recommendations

The present work can be brought forward to include more detailed follow-up studies especially on controller designs that aim at reducing fatigue damage within the drivetrain. Some possible recommendations for future works are presented in the following list:

- Since the performance of the energy shaping controller introducing negative damping at 1P (or rotor) frequency, effort should be made to reduce its effect.
- Control strategy to take into account the PI controller implemented on the drivetrain to simulate external torque and explore the additional possibility for load mitigation.
- Performance evaluations covering a wider spectrum of load cases including wave/wind directional probabilities instead of unidirectional to be carried out for a more complete analysis. With more simulations being done, optimization can be done to obtain an optimal set of control parameters.
- As the current work only looks into controller performance at above rated wind speed, it would be beneficial to look into the control regions below rated wind speed (more specifically region 2) to improve the overall performance of the controller.
- It is observed that although the implementation of IPC successfully reduced the pitching and yawing moment fluctuation he rotor plane, a mean moment offset is introduced. Studies can be made to investigate the phenomenon and whether the load on gears and bearings can be reduced with the removal of said moment.
- Effort can be made to filter out high frequency noise and extreme values from the simulation time series obtain from SIMPACK for better frequency response analysis of the drivetrain.
- blade root fatigue M-N curve has to be evaluated further for proper application

Bibliography

[1] M. A. Lackner. Controlling platform motions and reducing blade loads for floating wind turbines. *Wind Engineering*, 33(6), 2007.

[2] M. D. Pedersen. *Stabilization of Floating Wind Turbines*. PhD thesis, Norwegian University of Science and Technology, 2017.

[3] M. A. Lackner and G. Van Kuik. A comparison of smart rotor control approaches using trailing edge flaps and individual pitch control. In *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, page 685, Orlando, USA, 2009.

[4] E. Bossanyi. Individual blade pitch control for load reduction. *Wind Energy*, 6(DOI:10.1002/we.76):119–128, 2003.

[5] J. M. Jonkman. *Definition of the Floating System for Phase IV of OC3*. NREL, 2010.

[6] J. M. Jonkman, W. M. Butterfield, and G. Scott. *Definition of a 5-mw wind turbine for offshore system development", National Renewable Energy Laboratory*. NREL, 2009.

[7] A. R. Nejad, Y. Guo, Z. Gao, and T. Moan. Development of a 5 mw reference gearbox for offshore wind turbines. *Wind Energy*, 19:1089–1106, 2016.

[8] F. G. Nielsen and B. Skaare. Blade pitch control in a wind turbine installation, 2013. U.S. Patent 8,487,464 B2.

[9] D. Van-Nguyen and B. Biswajit. On the modeling of spar-type floating offshore wind turbines. *Key Engineering Materials*, 569-570, 2013.

[10] M. O. L. Hansen. *Aerodynamics of Wind Turbines, Second Edition*. Earthscan, 2008.

[11] E. Bossanyi, T. Burton, N. Jenkins, and D. Sharpe. *Wind Energy Handbook*. John Wiley and Sons, 2010.

[12] NTNU. Centre for ships and ocean structures annual report 2009, 2009. Norwegian University of Science and Technology.

[13] World Wind Energy Association. Wind power capacity worldwide reaches 600 gw, 53,9 gw added in 2018, February 2019.

[14] Wind Energy Today. Wind delivers the energy society wants, February 2019.

[15] J Dugstad. Global offshore wind market report 2018, 2018. Norwegian Energy Partners.

[16] International Electrotechnical Commission. *Wind energy generation systems - Part 3-1: Design requirements for fixed offshore wind turbines*, 1 edition, April 2019.

[17] E. Bossanyi. The design of closed loop controllers forwind turbines. *Wind Energy*, 3(DOI: 10.1002/we.34):149–163, 2000.

[18] F. G. Nielsen, T. D. Hanson, and B. Skaare. Integrated dynamic analysis of floating offshore wind turbines. In *25th International Conference on Offshore Mechanics and Arctic Engineering*, pages 671–679, Hamburg, Germany, 2006.

[19] T. J. Larsen and T. D. Hanson. A method to avoid negative damped low frequent tower vibrations for a floating, pitch controlled wind turbine. *Journal of Physics: Conference Series*, 75(012037), 2007.

[20] J. M. Jonkman. *Influence of control on the pitch damping of a floating wind turbine*. NREL, 2008.

[21] P. A. Fleming, A. Peiffer, and D. Schlipf. Wind turbine controller to mitigate structural loads on a floating wind turbine platform. *Journal of Offshore Mechanics and Arctic Engineering*, 141(061901-1), March 2019.

[22] A. R. Nejad, E. E. Bachynski, M. I. Kvittem, C. Luan, Z. Gao, and T. Moan. Stochastic dynamic load effect and fatigue damage analysis of drivetrains in land-based and tlp, spar and semi-submersible floating wind turbines. *Marine Structures*, 42:137–153, 2015.

[23] I. P. Girsang, J. S. Dhupia, E. Muljadi, and Singh M. Modeling and control to mitigate resonant load in variable-speed wind turbine drivetrain. *Journal of Emerging and Selected Topics in Power Electronics*, 1(4), 2013.

[24] International Electrotechnical Commission. *Wind energy generation systems - Part 3-2: Design requirements for floating offshore wind turbines*, 1 edition, April 2019.

[25] Norwegian Marine Technology Research Institute. *SIMO - Theory Manual Version 3.6, rev: 2*. Norwegian Marine Technology Research Institute, 2009.

[26] SINTEF Ocean. *RIFLEX 4.12.2 Theory Manual*. SINTEF Ocean, 2008.

[27] O. A. Bauchau. *Flexible Multibody Dynamics*. Springer, 2010.

[28] Y. Xing and T. Moan. Multibody modelling and analysis of a planet carrier in a wind turbine gearbox. *Wind Energy*, 16:1067–1089, 2013.

[29] Xing. Y, M. Karimirad, and T. Moan. Modelling and analysis of floating spar-type wind turbine drivetrain. *Wind Energy*, 14:565–587, 2014.

[30] J. M. Jonkman and M. L. Buhl. *TurbSim Users Guide for Version 1.40*. NREL, 2008.

[31] International Electrotechnical Commission. *Wind energy generation systems - Part 1: Design requirements*, 4 edition, February 2019.

[32] Dassault Systemes Simulia Corp. *Simpack Documentation*. Dassault Systemes Simulia Corp., 2016.

[33] Det Norske Veritas. *DNV-RP-C203: Fatigue Design of Offshore Steel Structures*. Det Norske Veritas, 2011.

[34] ISO. *Calculation of load capacity of spur and helical gears - Part 3: Calculation of tooth bending strength*, 2 edition, April 2007.

[35] G. Freebury and W. D. Musial. *Determining Equivalent Damage Loading for Full-Scale Wind Turbine Blade Fatigue Tests*. NREL, 2000.

[36] C. Bottasso, F. Campagnolo, A. Croce, and C. Tibaldi. Optimization-based study of bendtwist coupled rotor blades for passive and integrated passive/active load alleviation. *Wind Energy*, 16(8):1149–66, 2013.

[37] S. K. Chakrabarti. The theory and practice of hydrodynamics and vibration. *Advanced Series on Ocean Engineering*, 20, 2002.

[38] R. Coleman and A. Feingold. Theory of self-excited mechanical oscillations of helicopter rotors with hinged blades, 1958. NASA.

Appendix

A. Code for active damping controller

```
1
2  package no.marintek.wind.control;
3
4  import java.util.logging.Level;
5  import java.util.logging.Logger;
6  import no.marintek.wind.control.Feedback;
7  import no.marintek.wind.control.IController;
8  import no.marintek.wind.control.Measurements;
9
10 import java.io.*;
11 /*************************************************************************
12  *
13  *   Implementation of a conventional power wind turbine control system to
14  *   illustrate use of interfacing between RIFLEX and external controller
15  *
16  *   @version: 1.0
17  *   @author Lasse Bjermeland/ Erin Bachynski
18  *   @version: 1.1 modified by ChernFong Lee
19  *
20  *
21  * ************************************************************************/
22 public class WindTurbineController implements IController {
23 /*mychange*/
24     private int nblades,nstep;
25         private double simTime, dt;
26
27
28   /* Assigns default values, but these are all reset when the
29          * input file is read. */
30         private String torqueString = "TORQUE";
31         private String powerString = "POWER";
32         private int ConstPower = 0;
33
34
35         private double cornerFreq = 0.25; // corner frequency in Hz
36         private double cutoffFreq = 0.08; // cutoff frequency in Hz
37         private double kK = 0.109965; /*  Pitch angle where the the derivative
38          * of the aerodynamic power w.r.t. pitch has increased by a factor of
39          * two relative to the derivative at rated pitch (zero), rad. */
40         private double kI = 0.008068634; /* integral gain at min
41          * pitch setting (s)*/
42         private double kP = 0.01882681; /*proportional gain at
43          * min pitch setting */
44     private double kTP = -0.375; /*proportional gain for varying
45      * generator speed (1/(rad/s)) */
46
47
48         // PID control test variables
49         //private double kD = 0.0;
50         //private double oldPerr = 0.0;
51         private double newPerr = 0.0;
52
53         // generator torque variables
54                 private double VS_CtInSp  = 70.16224; /*Transitional generator speed
55          * (HSS side) between regions 1 and 1 1/2, rad/s */
56         private double VS_Rgn2K = 2.332287; /* Generator torque constant
57          * in Region 2 (HSS side), N-m/(rad/s)^2*/
58         private double VS_Rgn2Sp = 91.21091; /* Transitional generator speed
59          * (HSS side) between regions 1 1/2 and 2, rad/s*/
60         private double VS_SlPc = 10;  /*generator slip percentage
61          * in region 2.5 */
62         private double VS_Rgn3MP = 1; /*deg (Minimum pitch angle at which the
63          * torque is computed as if we are in region 3 regardless of
64          * the generator speed) */
65         private double Pnom = 5296600 ;  /*rated mechanical power */
66         private double Tnom = 43093.55;  /*constant torque in region 3 */
67         private double Ng = 97; /* Ng:1 gearbox ratio*/
68         private double omega_g_nom = 122.9096; /* reference HSS speed, rad/s*/
69         private double Qgmax = 47356; /*Qgmax, Maximum generator torque in
```

```
         * Region 3 (HSS side), N-m. */
        private double Qgmin = 0;/**/
        private double dQgmax = 15000; /*maximum generator torque rate Nm/s */
        private double dQgmin = -15000; /* Nm/s */

        // blade pitch limits
        private double th_max = 90 ;/*maximum blade pitch setting deg */
        private double th_min = 0; /* */
        private double dthmax = 8;/*maximum absolute blade pitch rate deg/s */
        private double dthmin = -8; /* */

        private double dTcontroller = 0.0125;
        private double lastControlTime = 0.0;
        private double ElapsedTime = 0.0;
        private double initialPitch = 0;
        private double warmUpTime = 0;
        private int endWarmUp = 0;


        /* Some variables for the default NREL controller */
        private double oldTorque, oldIntErr, oldOmegaGf;
        private double newTorque, newIntErr, newOmegaGf;
        private double[] measPitch;
        private double[] oldPitchC;
        private double[] newPitchC;

        private double dPitch, dTorque;

 //      output file
        private int writeOutputFile = 0;
        private int writePos = 0;
        private int writeVel = 0;
        private int writeAcc = 0;
        BufferedWriter out;
        FileWriter fstream;
        private double dTwrite = 0.0125;
        private double lastWriteTime = 0.0;
        private double ElapsedTimeW = 0.0;

//      fixed pitch option
        private int fixedPitchOn = 0;
        private double fixedPitch = 0.0;

        private double oldSurgeVel;
        private double newSurgeVel;

    /**
     * Initialize controller, called at startup
     *
     * @param dt Time step
     * @param nblades Number of blades
     * @param filename path to configuration file
     */
    @Override
    public void init(double dt, int nblades, String filename) {
                FileInputStream fis = null;
                BufferedInputStream bis = null;
                DataInputStream dis = null;

                this.simTime = 0.0;
        this.nblades = nblades;
                this.dt = dt;
                this.nstep = 0;
                this.oldTorque = 0;
                this.oldIntErr = 0;
                this.oldOmegaGf = 0;
                this.newSurgeVel = 0;
                this.oldSurgeVel = 0;
                this.newTorque = 0;
                this.newIntErr = 0;
                this.newOmegaGf = 0;


                this.measPitch = new double[this.nblades];
                this.oldPitchC = new double[this.nblades];
                this.newPitchC = new double[this.nblades];


        /* Read the config file*/

        BufferedReader inputStream = null;
        PrintWriter outputStream = null;
        int count = 0;

          try {
                inputStream =  new BufferedReader(new FileReader(filename));

            String lineString;
            System.out.println(" ");
            System.out.println("Control System Input ECHO:");
```

```
160             System.out.println(" ");
161             while ((lineString = inputStream.readLine()) != null)
162             {
163             //System.out.println(lineString);
164             int aposind = -1;
165             aposind = lineString.indexOf('\'') ;
166             if (aposind != -1)
167                 {
168             // this is a comment line, skip it
169                 }
170                 else { // not a comment!
171                 count = count +1;
172
173                 if (count == 1)
174                 {
175                     // set torque or power by reading power line
176                     int ConstPowerX = lineString.compareToIgnoreCase(powerString);
177                     if (ConstPowerX == 0) // that means that it is constant power
178                     {
179                         this.ConstPower = 1;
180                         System.out.println("Control with constant POWER in region 3");
181                     }
182                     else
183                     {
184                         ConstPowerX = lineString.compareToIgnoreCase(torqueString);
185                         if (ConstPowerX != 0)
186                         {
187                             System.out.println("Error reading POWER or TORQUE. Assuming POWER.");
188                             this.ConstPower = 1;
189                         }
190                         else
191                         {
192                             System.out.println("Control with constant TORQUE in region 3");
193                         }
194
195                     }
196
197                 }
198                 if (count == 2)
199                 {
200                     // corner frequency
201                     this.cornerFreq = Double.parseDouble(lineString);
202                    System.out.println("Corner Freq:       " + this.cornerFreq + " Hz");
203
204                 }
205                 if (count == 3)
206                 {
207                    this.kK = Double.parseDouble(lineString);
208                    System.out.println("Kk:                " + this.kK + " rad");
209                 }
210                  if (count == 4)
211                 {
212                    this.kI = Double.parseDouble(lineString);
213                    System.out.println("Ki:                " + this.kI + " ");
214                 }
215                  if (count == 5)
216                 {
217                    this.kP = Double.parseDouble(lineString);
218                    System.out.println("Kp:                " + this.kP + " s ");
219                 }
220                   if (count == 6)
221                 {
222                    this.VS_CtInSp = Double.parseDouble(lineString);
223                    System.out.println("VS_CtInSp:         " + this.VS_CtInSp + " rad/s");
224                 }
225                   if (count == 7)
226                 {
227                    this.VS_Rgn2K = Double.parseDouble(lineString);
228                    System.out.println("VS_Rgn2K:          " + this.VS_Rgn2K + " N-m/(rad/s)^2");
229                 }
230                   if (count == 8)
231                 {
232                    this.VS_Rgn2Sp = Double.parseDouble(lineString);
233                    System.out.println("VS_Rgn2Sp:         " + this.VS_Rgn2Sp + " rad/s");
234                 }
235                   if (count == 9)
236                 {
237                    this.VS_SlPc = Double.parseDouble(lineString);
238                    System.out.println("VS_SlPc:           " + this.VS_SlPc + " % ");
239                 }
240                   if (count == 10)
241                 {
242                    this.VS_Rgn3MP = Double.parseDouble(lineString);
243                    System.out.println("VS_Rgn3MP:         " + this.VS_Rgn3MP + " deg ");
244                 }
245                   if (count == 11)
246                 {
247                    this.Pnom = Double.parseDouble(lineString);
248                    System.out.println("Pnom:              " + this.Pnom + " W ");
249                 }
```

```
                if (count == 12)
               {
                  this.Tnom = Double.parseDouble(lineString);
                  System.out.println("Tnom:              " + this.Tnom + " Nm ");
               }
                if (count == 13)
               {
                  this.Ng = Double.parseDouble(lineString);
                  System.out.println("Ng:                " + this.Ng + ":1 ");
               }
                if (count == 14)
               {
                  this.omega_g_nom = Double.parseDouble(lineString);
                  System.out.println("omega_g_nom:       " + this.omega_g_nom + " rad/s ");
               }
                if (count == 15)
               {
                  this.Qgmax = Double.parseDouble(lineString);
                  System.out.println("Qgmax:             " + this.Qgmax + " Nm ");
               }
                if (count == 16)
               {
                  this.Qgmin = Double.parseDouble(lineString);
                  System.out.println("Qgmin:             " + this.Qgmin + " Nm ");
               }
                if (count == 17)
               {
                  this.dQgmax = Double.parseDouble(lineString);
                  System.out.println("dQgmax:            " + this.dQgmax + " Nm/s ");
               }
                if (count == 18)
               {
                  this.dQgmin = Double.parseDouble(lineString);
                  System.out.println("dQgmin:            " + this.dQgmin + " Nm/s ");
               }
                if (count == 19)
               {
                  this.th_max = Double.parseDouble(lineString);
                  System.out.println("th_max:            " + this.th_max + " deg ");
               }
                if (count == 20)
               {
                  this.th_min = Double.parseDouble(lineString);
                  System.out.println("th_min:            " + this.th_min + " deg ");
               }
                if (count == 21)
               {
                  this.dthmax = Double.parseDouble(lineString);
                  System.out.println("dthmax:            " + this.dthmax + " deg/s ");
               }
                if (count == 22)
               {
                  this.dthmin = Double.parseDouble(lineString);
                  System.out.println("dthmin:            " + this.dthmin + " deg/s ");
               }
                if (count == 23)
               {
                  this.dTcontroller = Double.parseDouble(lineString);
                  System.out.println("Control TimeStep:           " + this.dTcontroller + " s ");
               }
          /* ********* WARM-UP INPUT ********** */
                 if (count == 24)
               {
                    this.initialPitch = Double.parseDouble(lineString);
                    System.out.println("Initial blade pitch: " + this.initialPitch + " deg" );
               }
                if (count == 25)
               {
                    this.warmUpTime = Double.parseDouble(lineString);
                    System.out.println("Warm-up time: " + this.warmUpTime + "s" );
               }
           /* ********* INPUT FOR OUTPUT FILE ********** */
                if (count == 26)
               {
                   this.writeOutputFile = Integer.parseInt(lineString);
                   if (this.writeOutputFile == 1)
                   {
                   System.out.println("Output file ControlOutput.txt will be created");
                   }
                   else
                   {
                   System.out.println("No control output file will be created");
                   }
               }
                if (count == 27)
                {
                    this.writePos = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writePos == 1))
                    {
                        System.out.println("Position output included (m-rad)");
```

```
                    }
                }
                if (count == 28)
                {
                    this.writeVel = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writeVel == 1))
                    {
                        System.out.println("Velocity output included (m/s - rad/s)");
                    }
                }
                if (count == 29)
                {
                    this.writeAcc = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writeAcc == 1))
                    {
                        System.out.println("Acceleration output included (m/s^2 - rad/s^2)");
                    }
                }
                if (count == 30)
              {
                    this.dTwrite = Double.parseDouble(lineString);
                    if  (this.writeOutputFile == 1)
                    {
                    System.out.println("Control output time interval: " + this.dTwrite + "s" );
                    }
              }
                if (count == 31)
              {
                    this.fixedPitchOn = Integer.parseInt(lineString);
                    if  (this.fixedPitchOn == 1)
                    {
                    System.out.println("Fixed Pitch Option On " );
                    }
              }
                 if (count == 32)
              {
                    this.fixedPitch = Double.parseDouble(lineString);
                    if  (this.fixedPitchOn == 1)
                    {
                    System.out.println("Fixed Pitch Angle: " + this.fixedPitch + "deg" );
                    }
              }

              } // end else (not comment)

          } // end while
           System.out.println(" ");
          } // end try
          catch (FileNotFoundException e) {
          e.printStackTrace();
          }
          catch (IOException e) {
          e.printStackTrace();
      } // end of try reading config file
    finally {
          if (inputStream != null) {
              try {
                  inputStream.close();
              } catch (IOException ex) {
                  Logger.getLogger(WindTurbineController.class.getName())
                          .log(Level.SEVERE, null, ex);
              }
          }

        }


        // Fix units for some input
        this.cornerFreq = this.cornerFreq*2*Math.PI;
        this.th_max = this.th_max*Math.PI/180.0;
        this.th_min = this.th_min*Math.PI/180.0;
        this.dthmax = this.dthmax*Math.PI/180.0;
        this.dthmin = this.dthmin*Math.PI/180.0;
        this.initialPitch = this.initialPitch*Math.PI/180.0;
        this.fixedPitch = this.fixedPitch*Math.PI/180.0;
        this.VS_Rgn3MP = this.VS_Rgn3MP*Math.PI/180;

          System.out.println("Controller successfully initialized");
          System.out.println(" ");

          // Output file
          if (this.writeOutputFile == 1)
          {
            try
            {
            this.fstream = new FileWriter("ControlOutput.txt");
            this.out = new BufferedWriter(fstream);
            }
               catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
```

```
            }
          }

    } // end of init


 /*************************************************************************/
 /*************************************************************************/
    /**
     * Called each time step
     *
     * @param measurements: Measurements from RIFLEX used to calculate
       * controller actions
     *
     *    Measurements Contains:
     *
     *    omega:         Rotor velocity in rad/s
     *    pitch angles:  Blade pitch angles in radians
     *    position       (x,y,z,rx,ry,rz) position array, angles in radians
     *    velocity       (vx,vy,vz,vrx,vry,vrz) velocity array, angular
       *                                                  velocity in rad/s
     *    acceleration   (ax,ay,az,arx,ary,arz) acceleration array,
       *                                  angular acceleration in rad/s^2
     *
     * @param feedback: Control feedback to RIFLEX
     *
     *   Feedback Contains:
     *
     *   Torque reference:        Actuator torque to apply on rotor axis
     *   Pitch angle references:  Controller pitch angle in radians
     *
     *   Used for presentation only:
     *   gear shaft omega:        Gear shaft rotor speed (rpm)
     *   generated power:         Generated electrical power (kW)
     *
     */
    @Override
    public void step(Measurements measurements, Feedback feedback)
        {

                // figure out our current simulation time
                double omega_g = measurements.getOmega()*this.Ng; // HSS side, rad/s
                //double pitch1 = measurements.getPitchAngle(0); // rad
                for (int i = 0; i < nblades; i++)
                {
                    this.measPitch[i] = measurements.getPitchAngle(i);
                }

                this.nstep = this.nstep + 1;
                this.simTime =  (this.nstep-1)*this.dt;
                double[] pos = measurements.getPosition();
                double[] vel = measurements.getVelocity();
                double[] accel = measurements.getAcceleration();



                // write the measurements if enough time has passed
                if (this.writeOutputFile == 1)
                {
                    this.ElapsedTimeW = this.simTime - this.lastWriteTime;
                    if ( this.ElapsedTimeW>this.dTwrite )
                    {
                    try
                        {

                      this.out.write("" + simTime );
                      if (this.writePos == 1)
                      {
                      this.out.write(" " + pos[0] + " " + pos[1] + " " + pos[2]
                              + " " + pos[3] + " " + pos[4] + " " + pos[5] );
                      this.out.write(" " +  measPitch[0] + " " + measPitch[1] + " " + measPitch[2] );
                      }
                      if (this.writeVel == 1)
                      {
                      this.out.write(" " +  vel[0] + " " + vel[1] + " " + vel[2]
                              + " " + vel[3] + " " + vel[4] + " " + vel[5]  );
                      }
                      if (this.writeAcc == 1)
                      {
                      this.out.write(" " +  accel[0] + " " + accel[1] + " " + accel[2]
                              + " " + accel[3] + " " + accel[4] + " " + accel[5]  );
                      }

                      this.out.newLine();
                      this.out.flush();

                         }
                         catch (Exception e){//Catch exception if any
                        System.err.println("Error: " + e.getMessage());
                        }
```

```
                    this.lastWriteTime = this.simTime;
                     }
                }



                // send some results to Riflex
                feedback.setGearShaftOmega(this.newOmegaGf); // fixed in V3.7.23
                feedback.setGeneratedPower(measurements.getOmega()*this.newTorque/1000*this.Ng); // kW

                // only update the control actions if enough time has passed
                this.ElapsedTime = this.simTime - this.lastControlTime;
                if ( this.dTcontroller<=this.ElapsedTime*1.00001 )
                {
                    normalControl(omega_g,vel[0]); // call normal control routine (modificationCF)
                    this.lastControlTime = this.simTime;
                }
                 // send the commands to Riflex
                for (int i = 0; i < nblades; i++)
                {
                    feedback.setPitchAngleReference(i, this.newPitchC[i]);
                }
                feedback.setTorqueReference( this.newTorque/1000*this.Ng);


                // Options other than normal control
                // warm-up with fixed initial pitch
               if (this.simTime < this.warmUpTime)
               {
                   double WUpitch = 0;
                   if(this.simTime < this.warmUpTime/2)
                   {
                   WUpitch = this.initialPitch*this.simTime/(this.warmUpTime/2);
                   }

                   else
                   {
                   WUpitch = this.initialPitch;
                   }

                   WUpitch = Math.min(Math.max(WUpitch,this.th_min),this.th_max);
                   this.dPitch = Math.min( Math.max((WUpitch-this.measPitch[0])/this.ElapsedTime,this.dthmin)
                     ,this.dthmax);
                   WUpitch = this.measPitch[0] + this.dPitch*this.ElapsedTime;
                   WUpitch = Math.min(Math.max(WUpitch,this.th_min),this.th_max);

                    for (int j = 0; j < nblades; j++)
                         {
                         feedback.setPitchAngleReference(j,WUpitch);
                         this.oldPitchC[j] = WUpitch;
                         this.newPitchC[j] = WUpitch;
                         }
                }
               else
               { // fixed pitch
                   if (this.fixedPitchOn == 1)
                   {
                     for (int j = 0; j < nblades; j++)
                        {
                        feedback.setPitchAngleReference(j,this.fixedPitch);
                        this.oldPitchC[j] = this.fixedPitch;
                        this.newPitchC[j] = this.fixedPitch;
                        this.dPitch = 0.0;
                        }

                   }


                }



        }

/*********************************************************************/
/* Normal control routine for pitch*/
        private void normalControl(double omega_g, double SurgeVel) //modificationCF
        {
                        // filter the generator speed
            double Alpha = Math.exp(-1*this.ElapsedTime*this.cornerFreq );
            this.newOmegaGf = (1.0-Alpha)*omega_g +
                    Alpha*this.oldOmegaGf;

                        // filter the surge velocity input
            double Alpha2 = Math.exp(-1*this.ElapsedTime*this.cutoffFreq);
            this.newSurgeVel = (1.0-Alpha2)*SurgeVel +
                    Alpha2*this.oldSurgeVel;

            // Computer other torque control parameters
```

```
            // modificationCF
            double omega_g_var = this.omega_g_nom*(1+kTP*this.newSurgeVel);
            double VS_RtGnSp = 0.99*omega_g_var;
            // double VS_RtGnSp = 0.99*this.omega_g_nom;
            double VS_SySp = VS_RtGnSp/(1+0.01*this.VS_SlPc);
            double VS_Slope15=(this.VS_Rgn2K*this.VS_Rgn2Sp*this.VS_Rgn2Sp)/
                    (this.VS_Rgn2Sp - this.VS_CtInSp);
            double VS_Slope25 = (this.Pnom/VS_RtGnSp)/(VS_RtGnSp-VS_SySp);
            double VS_TrGnSp = 0;
            if (this.VS_Rgn2K == 0)
            {
                VS_TrGnSp = VS_SySp;
            }
            else
            {
                VS_TrGnSp = (VS_Slope25 - Math.sqrt(VS_Slope25*
                (VS_Slope25-4.0*this.VS_Rgn2K*VS_SySp)))/
                        (2*this.VS_Rgn2K);
            }

/*******************************************************************************/
            // generator torque control
            if ((this.newOmegaGf >= VS_RtGnSp) ||
                   ((this.simTime>=this.warmUpTime) &  ( ( Math.abs(this.oldPitchC[0]) >= this.VS_Rgn3MP) ) & this.
                           fixedPitchOn == 0))
            { // region 3
                if (this.ConstPower == 1)
                {  // constant power
                this.newTorque = this.Pnom/this.newOmegaGf;
                }
                else
                { // constant torque
                this.newTorque = this.Tnom;
                }
            }
            else if (this.newOmegaGf <= this.VS_CtInSp)
            { // region 1
                this.newTorque = 0.0;
            }
            else if (this.newOmegaGf < this.VS_Rgn2Sp)
            { // region 1.5
                this.newTorque = VS_Slope15*
                        (this.newOmegaGf - this.VS_CtInSp);
            }
            else if (this.newOmegaGf < VS_TrGnSp)
            { // region 2
                this.newTorque = this.VS_Rgn2K*
                        Math.pow(this.newOmegaGf,2);
            }
            else
            { // region 2.5
                this.newTorque = VS_Slope25*(this.newOmegaGf - VS_SySp);
            }

/*******************************************************************************/
            // PI controller for pitch angle

            // Speed error
            // modificationCF
            double Ep = this.newOmegaGf - omega_g_var;
            //double Ep = this.newOmegaGf - this.omega_g_nom;
            this.newPerr = Ep;

            // integrated speed error
            double Ei = this.oldIntErr + Ep*this.ElapsedTime;


             // gain scheduling correction factor
            double GK = 1.0/(1.0 + this.oldPitchC[0]/this.kK ); //note pitch1 is in rad!


            if ((this.endWarmUp == 0) & (this.simTime >= this.warmUpTime))
            {
                Ei = this.initialPitch/this.kI/GK;
                this.oldIntErr = Ei;
                this.newIntErr = Ei;
                System.out.println("End of controller warm-up");
                this.endWarmUp = 1;
            }


            // saturate integral term
            Ei = Math.min(Math.max(Ei,this.th_min/(GK*this.kI)),this.th_max/(GK*this.kI));


           // compute pitch commands
            double PitComP = GK*this.kP*Ep;
            double PitComI = GK*this.kI*Ei;
```

```
            double PitComT = PitComP + PitComI;
            this.newIntErr = Ei;

/*************************************************************************/
            // saturation limits of control signals
            this.newTorque =
                  Math.min(Math.max(this.newTorque,this.Qgmin),this.Qgmax);
            double dTorque =
                    Math.min(
                    Math.max((this.newTorque-this.oldTorque)/this.ElapsedTime,this.dQgmin)
                    ,this.dQgmax);
            this.newTorque = this.oldTorque + dTorque*this.ElapsedTime;

            PitComT =
                  Math.min(Math.max(PitComT,this.th_min),this.th_max);
            for (int i = 0; i < nblades; i++)
                {
                 double ddes = (PitComT-this.measPitch[i])/this.ElapsedTime;
                 dPitch = Math.min( Math.max(ddes,this.dthmin)
                    ,this.dthmax);
                this.newPitchC[i] = this.oldPitchC[i] + dPitch*this.ElapsedTime; //dPitch*this.dTcontroller;//
                this.oldPitchC[i] = this.newPitchC[i];
                }


/*************************************************************************/
            // set the variables for the next call

            this.oldIntErr = this.newIntErr;
            this.oldTorque = this.newTorque;
            this.oldOmegaGf = this.newOmegaGf;
            this.oldSurgeVel = this.newSurgeVel;

        }

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

    /**
     * Called once after simulation is done
     */
    @Override
    public void finish()
        {
            if (this.writeOutputFile == 1)
            {
         try
                {
                   this.out.flush();
                   this.out.close();
                }
             catch (Exception e)
                {//Catch exception if any
                   System.err.println("Error: " + e.getMessage());
                }
            }


    }

}
```

B. Code for energy shaping controller

```
package no.marintek.wind.control;

import java.util.logging.Level;
import java.util.logging.Logger;
import no.marintek.wind.control.Feedback;
import no.marintek.wind.control.IController;
import no.marintek.wind.control.Measurements;

import java.io.*;
/*************************************************************************
 *
 *   Implementation of a conventional power wind turbine control system to
 *   illustrate use of interfacing between RIFLEX and external controller
 *
 *   @version: 1.0
 *   @author Lasse Bjermeland/ Erin Bachynski
 *   @version: 1.1 modified by ChernFong Lee
 *
 *
 * ***********************************************************************/
public class WindTurbineController implements IController {
```

```
/*mychange*/
    private int nblades,nstep;
        private double simTime, dt, Steptime;


  /* Assigns default values, but these are all reset when the
        * input file is read. */
        private String torqueString = "TORQUE";
        private String powerString = "POWER";
        private int ConstPower = 0;
        private int IPCon = 1;


        private double cornerFreq = 0.25; // corner frequency in Hz
        private double cutoffFreq = 0.08; // cutoff frequency in Hz
        private double halfFreq = 0.08;    // moment cutoff frequency in Hz
        private double kK = 0.109965; /*  Pitch angle where the the derivative
         * of the aerodynamic power w.r.t. pitch has increased by a factor of
         * two relative to the derivative at rated pitch (zero), rad. */
        private double kI = 0.008068634; /* integral gain at min
         * pitch setting (s)*/
        private double kP = 0.01882681; /*proportional gain at
         * min pitch setting */
    private double kTP = -0.125; /*proportional gain for varying
     * generator speed (1/(rad/s)) */
        private double kP_ipc = 0.06; /*proportional gain for IPC
        rad/(10^6Nm) */
        private double kI_ipc = 0.01; /*integral gain for IPC
        rad/s/(10^6Nm) */
        private double kD_ipc = -0.001; /*derivative gain for IPC
        rad*s/(10^6Nm) */


        // PID control test variables
        //private double kD = 0.0;
        //private double oldPerr = 0.0;
        private double newPerr = 0.0;

        // generator torque variables
    private double VS_CtInSp  = 70.16224; /*Transitional generator speed
         * (HSS side) between regions 1 and 1 1/2, rad/s */
        private double VS_Rgn2K = 2.332287; /* Generator torque constant
         * in Region 2 (HSS side), N-m/(rad/s)^2*/
        private double VS_Rgn2Sp = 91.21091; /* Transitional generator speed
         * (HSS side) between regions 1 1/2 and 2, rad/s*/
        private double VS_SlPc = 10;  /*generator slip percentage
         * in region 2.5 */
        private double VS_Rgn3MP = 1; /*deg (Minimum pitch angle at which the
         * torque is computed as if we are in region 3 regardless of
         * the generator speed) */
        private double Pnom = 5296600 ;  /*rated mechanical power */
        private double Tnom = 43093.55;  /*constant torque in region 3 */
        private double Ng = 97; /* Ng:1 gearbox ratio*/
        private double omega_g_nom = 122.9096; /* reference HSS speed, rad/s*/
        private double Qgmax = 47356; /*Qgmax, Maximum generator torque in
         * Region 3 (HSS side), N-m. */
        private double Qgmin = 0;/**/
        private double dQgmax = 15000; /*maximum generator torque rate Nm/s */
        private double dQgmin = -15000; /* Nm/s */

        // blade pitch limits
        private double th_max = 90 ;/*maximum blade pitch setting deg */
        private double th_min = 0; /* */
        private double dthmax = 8;/*maximum absolute blade pitch rate deg/s */
        private double dthmin = -8; /* */

        private double dTcontroller = 0.0125;
        private double lastControlTime = 0.0;
        private double ElapsedTime = 0.0;
        private double initialPitch = 0;
        private double warmUpTime = 0;
        private int endWarmUp = 0;

        // blade initial twist
        private double twist = 0;

        /* Some variables for the default NREL controller */
        private double oldTorque, oldIntErr, oldOmegaGf;
        private double newTorque, newIntErr, newOmegaGf;
        private double[] measPitch;
        private double[] measBMY;
        private double[] measBMZ;
        private double[] oldmeasBMY;
        private double[] oldmeasBMZ;
        private double[] oldAzi;
        private double[] measPsi;
        private double[] ipcPitch;
        private double[] oldPitchC;
        private double[] newPitchC;
```

```
873         private double dPitch, dTorque;
874 
875  //       output file
876         private int writeOutputFile = 0;
877         private int writePos = 0;
878         private int writeVel = 0;
879         private int writeAcc = 0;
880         BufferedWriter out;
881         FileWriter fstream;
882         private double dTwrite = 0.0125;
883         private double lastWriteTime = 0.0;
884         private double ElapsedTimeW = 0.0;
885 
886  //     fixed pitch option
887         private int fixedPitchOn = 0;
888         private double fixedPitch = 0.0;
889 
890  //     log surge vel and for k and k+1
891         private double oldSurgeVel;
892         private double newSurgeVel;
893         private double oldOmega_g_var;
894         private double deltaomega_g_var;
895         private double omega_g_var;
896         private double My2cm;
897         private double My3cm;
898         private double oldIntMy2cm;
899         private double oldMy2cm;
900         private double oldIntMy3cm;
901         private double oldMy3cm;
902 
903     /**
904      * Initialize controller, called at startup
905      *
906      * @param dt Time step
907      * @param nblades Number of blades
908      * @param filename path to configuration file
909      */
910     @Override
911     public void init(double dt, int nblades, String filename) {
912                 FileInputStream fis = null;
913                 BufferedInputStream bis = null;
914                 DataInputStream dis = null;
915 
916                 this.simTime = 0.0;
917         this.nblades = nblades;
918                 this.dt = dt;
919                 this.Steptime = 0.005;
920                 this.nstep = 0;
921                 this.oldTorque = 0;
922                 this.oldIntErr = 0;
923                 this.oldOmegaGf = 0;
924                 this.newTorque = 0;
925                 this.newIntErr = 0;
926                 this.newOmegaGf = 0;
927                 this.newSurgeVel = 0;
928                 this.oldSurgeVel = 0;
929                 this.oldOmega_g_var = this.omega_g_nom;
930                 this.deltaomega_g_var = 0;
931                 this.omega_g_var = this.omega_g_nom;
932                 this.My2cm = 0;
933                 this.My3cm = 0;
934                 this.oldIntMy2cm = 0;
935                 this.oldMy2cm = 0;
936                 this.oldIntMy3cm = 0;
937                 this.oldMy3cm = 0;
938 
939 
940                 this.measPitch = new double[this.nblades];
941                 this.measBMY = new double[this.nblades];
942                 this.measBMZ = new double[this.nblades];
943                 this.oldmeasBMY = new double[this.nblades];
944                 this.oldmeasBMZ = new double[this.nblades];
945                 this.oldAzi = new double[this.nblades];
946                 this.measPsi = new double[this.nblades];
947                 this.oldAzi[0] = -1*Math.PI/2;
948                 this.oldAzi[1] = -1*Math.PI/2 + 2*Math.PI/3;
949                 this.oldAzi[2] = -1*Math.PI/2 + 4*Math.PI/3;
950                 this.ipcPitch = new double[this.nblades];
951                 this.oldPitchC = new double[this.nblades];
952                 this.newPitchC = new double[this.nblades];
953 
954 
955         /* Read the config file*/
956 
957         BufferedReader inputStream = null;
958         PrintWriter outputStream = null;
959         int count = 0;
960 
961           try {
962                 inputStream =  new BufferedReader(new FileReader(filename));
```

```
963             String lineString;
964             System.out.println(" ");
965             System.out.println("Control System Input ECHO:");
966             System.out.println(" ");
967             while ((lineString = inputStream.readLine()) != null)
968             {
969             //System.out.println(lineString);
970             int aposind = -1;
971             aposind = lineString.indexOf('\'') ;
972             if (aposind != -1)
973                 {
974             // this is a comment line, skip it
975                 }
976                 else { // not a comment!
977                 count = count +1;
978
979                 if (count == 1)
980                 {
981                     // set torque or power by reading power line
982                     int ConstPowerX = lineString.compareToIgnoreCase(powerString);
983                     if (ConstPowerX == 0) // that means that it is constant power
984                     {
985                         this.ConstPower = 1;
986                         System.out.println("Control with constant POWER in region 3");
987                     }
988                     else
989                     {
990                         ConstPowerX = lineString.compareToIgnoreCase(torqueString);
991                         if (ConstPowerX != 0)
992                         {
993                             System.out.println("Error reading POWER or TORQUE. Assuming POWER.");
994                             this.ConstPower = 1;
995                         }
996                         else
997                         {
998                             System.out.println("Control with constant TORQUE in region 3");
999                         }
1000
1001                    }
1002
1003                }
1004                if (count == 2)
1005                {
1006                    // corner frequency
1007                    this.cornerFreq = Double.parseDouble(lineString);
1008                   System.out.println("Corner Freq:       " + this.cornerFreq + " Hz");
1009
1010                }
1011                if (count == 3)
1012                {
1013                   this.kK = Double.parseDouble(lineString);
1014                   System.out.println("Kk:                " + this.kK + " rad");
1015                }
1016                 if (count == 4)
1017                {
1018                   this.kI = Double.parseDouble(lineString);
1019                   System.out.println("Ki:                " + this.kI + " ");
1020                }
1021                 if (count == 5)
1022                {
1023                   this.kP = Double.parseDouble(lineString);
1024                   System.out.println("Kp:                " + this.kP + " s ");
1025                }
1026                  if (count == 6)
1027                {
1028                   this.VS_CtInSp = Double.parseDouble(lineString);
1029                   System.out.println("VS_CtInSp:         " + this.VS_CtInSp + " rad/s");
1030                }
1031                  if (count == 7)
1032                {
1033                   this.VS_Rgn2K = Double.parseDouble(lineString);
1034                   System.out.println("VS_Rgn2K:          " + this.VS_Rgn2K + " N-m/(rad/s)^2");
1035                }
1036                  if (count == 8)
1037                {
1038                   this.VS_Rgn2Sp = Double.parseDouble(lineString);
1039                   System.out.println("VS_Rgn2Sp:         " + this.VS_Rgn2Sp + " rad/s");
1040                }
1041                  if (count == 9)
1042                {
1043                   this.VS_SlPc = Double.parseDouble(lineString);
1044                   System.out.println("VS_SlPc:           " + this.VS_SlPc + " % ");
1045                }
1046                  if (count == 10)
1047                {
1048                   this.VS_Rgn3MP = Double.parseDouble(lineString);
1049                   System.out.println("VS_Rgn3MP:         " + this.VS_Rgn3MP + " deg ");
1050                }
1051                  if (count == 11)
1052
```

```
1053                {
1054                   this.Pnom = Double.parseDouble(lineString);
1055                   System.out.println("Pnom:              " + this.Pnom + " W ");
1056                }
1057                  if (count == 12)
1058                {
1059                   this.Tnom = Double.parseDouble(lineString);
1060                   System.out.println("Tnom:              " + this.Tnom + " Nm ");
1061                }
1062                  if (count == 13)
1063                {
1064                   this.Ng = Double.parseDouble(lineString);
1065                   System.out.println("Ng:                " + this.Ng + ":1 ");
1066                }
1067                  if (count == 14)
1068                {
1069                   this.omega_g_nom = Double.parseDouble(lineString);
1070                   System.out.println("omega_g_nom:       " + this.omega_g_nom + " rad/s ");
1071                }
1072                  if (count == 15)
1073                {
1074                   this.Qgmax = Double.parseDouble(lineString);
1075                   System.out.println("Qgmax:             " + this.Qgmax + " Nm ");
1076                }
1077                  if (count == 16)
1078                {
1079                   this.Qgmin = Double.parseDouble(lineString);
1080                   System.out.println("Qgmin:             " + this.Qgmin + " Nm ");
1081                }
1082                  if (count == 17)
1083                {
1084                   this.dQgmax = Double.parseDouble(lineString);
1085                   System.out.println("dQgmax:            " + this.dQgmax + " Nm/s ");
1086                }
1087                  if (count == 18)
1088                {
1089                   this.dQgmin = Double.parseDouble(lineString);
1090                   System.out.println("dQgmin:            " + this.dQgmin + " Nm/s ");
1091                }
1092                  if (count == 19)
1093                {
1094                   this.th_max = Double.parseDouble(lineString);
1095                   System.out.println("th_max:            " + this.th_max + " deg ");
1096                }
1097                  if (count == 20)
1098                {
1099                   this.th_min = Double.parseDouble(lineString);
1100                   System.out.println("th_min:            " + this.th_min + " deg ");
1101                }
1102                  if (count == 21)
1103                {
1104                   this.dthmax = Double.parseDouble(lineString);
1105                   System.out.println("dthmax:            " + this.dthmax + " deg/s ");
1106                }
1107                  if (count == 22)
1108                {
1109                   this.dthmin = Double.parseDouble(lineString);
1110                   System.out.println("dthmin:            " + this.dthmin + " deg/s ");
1111                }
1112                  if (count == 23)
1113                {
1114                   this.dTcontroller = Double.parseDouble(lineString);
1115                   System.out.println("Control TimeStep:           " + this.dTcontroller + " s ");
1116                }
1117             /* ********* WARM-UP INPUT *********** */
1118                   if (count == 24)
1119                {
1120                     this.initialPitch = Double.parseDouble(lineString);
1121                     System.out.println("Initial blade pitch: " + this.initialPitch + " deg" );
1122                }
1123                  if (count == 25)
1124                {
1125                     this.warmUpTime = Double.parseDouble(lineString);
1126                     System.out.println("Warm-up time: " + this.warmUpTime + "s" );
1127                }
1128              /* ********* INPUT FOR OUTPUT FILE *********** */
1129                   if (count == 26)
1130                {
1131                    this.writeOutputFile = Integer.parseInt(lineString);
1132                    if (this.writeOutputFile == 1)
1133                    {
1134                    System.out.println("Output file ControlOutput.txt will be created");
1135                    }
1136                    else
1137                    {
1138                    System.out.println("No control output file will be created");
1139                    }
1140                }
1141                  if (count == 27)
1142                  {
```

```
                    this.writePos = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writePos == 1))
                    {
                        System.out.println("Position output included (m-rad)");
                    }
                }
                if (count == 28)
                {
                    this.writeVel = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writeVel == 1))
                    {
                        System.out.println("Velocity output included (m/s - rad/s)");
                    }
                }
                if (count == 29)
                {
                    this.writeAcc = Integer.parseInt(lineString);
                    if ( (this.writeOutputFile == 1) && (this.writeAcc == 1))
                    {
                        System.out.println("Acceleration output included (m/s^2 - rad/s^2)");
                    }
                }
                if (count == 30)
              {
                   this.dTwrite = Double.parseDouble(lineString);
                   if  (this.writeOutputFile == 1)
                   {
                   System.out.println("Control output time interval: " + this.dTwrite + "s" );
                   }
              }
                if (count == 31)
              {
                   this.fixedPitchOn = Integer.parseInt(lineString);
                   if  (this.fixedPitchOn == 1)
                   {
                   System.out.println("Fixed Pitch Option On " );
                   }
              }
                 if (count == 32)
              {
                   this.fixedPitch = Double.parseDouble(lineString);
                   if  (this.fixedPitchOn == 1)
                   {
                   System.out.println("Fixed Pitch Angle: " + this.fixedPitch + "deg" );
                   }
              }

              } // end else (not comment)

          } // end while
           System.out.println(" ");
          } // end try
          catch (FileNotFoundException e) {
          e.printStackTrace();
          }
          catch (IOException e) {
          e.printStackTrace();
      } // end of try reading config file
  finally {
         if (inputStream != null) {
             try {
                 inputStream.close();
             } catch (IOException ex) {
                 Logger.getLogger(WindTurbineController.class.getName())
                         .log(Level.SEVERE, null, ex);
             }
         }

       }


         // Fix units for some input
         this.twist = this.twist*Math.PI/180.0;
         this.cornerFreq = this.cornerFreq*2*Math.PI;
         this.cutoffFreq = this.cutoffFreq*2*Math.PI;
         this.halfFreq = this.halfFreq*2*Math.PI;
         this.th_max = this.th_max*Math.PI/180.0;
         this.th_min = this.th_min*Math.PI/180.0;
         this.dthmax = this.dthmax*Math.PI/180.0;
         this.dthmin = this.dthmin*Math.PI/180.0;
         this.initialPitch = this.initialPitch*Math.PI/180.0;
         this.fixedPitch = this.fixedPitch*Math.PI/180.0;
         // CF modification 3 degrees instead of the default 1 in baseline
         this.VS_Rgn3MP = 3;
         this.VS_Rgn3MP = this.VS_Rgn3MP*Math.PI/180;

           System.out.println("Controller successfully initialized");
           System.out.println(" ");

           // Output file
```

```
            if (this.writeOutputFile == 1)
            {
              try
              {
              this.fstream = new FileWriter("ControlOutput.txt");
              this.out = new BufferedWriter(fstream);
              }
                 catch (Exception e){//Catch exception if any
              System.err.println("Error: " + e.getMessage());
              }
            }

    } // end of init


  /************************************************************************/
  /************************************************************************/
     /**
      * Called each time step
      *
      * @param measurements: Measurements from RIFLEX used to calculate
          * controller actions
      *
      *    Measurements Contains:
      *
      *    omega:         Rotor velocity in rad/s
      *    pitch angles:  Blade pitch angles in radians
      *    position       (x,y,z,rx,ry,rz) position array, angles in radians
      *    velocity       (vx,vy,vz,vrx,vry,vrz) velocity array, angular
          *                                                  velocity in rad/s
      *    acceleration   (ax,ay,az,arx,ary,arz) acceleration array,
          *                                    angular acceleration in rad/s^2
          *    blade root moments (BMY, BMZ)  in 10^3kNm
      *
      * @param feedback: Control feedback to RIFLEX
      *
      *   Feedback Contains:
      *
      *   Torque reference:        Actuator torque to apply on rotor axis
      *   Pitch angle references:  Controller pitch angle in radians
      *
      *   Used for presentation only:
      *   gear shaft omega:        Gear shaft rotor speed (rpm)
      *   generated power:         Generated electrical power (kW)
      *
      */
     @Override
     public void step(Measurements measurements, Feedback feedback)
         {

                 // figure out our current simulation time
                 double omega_g = measurements.getOmega()*this.Ng; // HSS side, rad/s
                 //double pitch1 = measurements.getPitchAngle(0); // rad
                 for (int i = 0; i < nblades; i++)
                 {
                     this.measPitch[i] = measurements.getPitchAngle(i);
                 }

                 // get flapwise bending moment and azimuthal angle of 3 blades
                 double[] BMY;
                 double[] BMZ;
                 BMY = new double[this.nblades];
                 BMZ = new double[this.nblades];

                 for (int i = 0; i < nblades; i++)
                 {
                     BMY[i] = measurements.getBladeRootBMY(i)/1000; // MNm
                     BMZ[i] = measurements.getBladeRootBMZ(i)/1000; // MNm
                 }

                 this.nstep = this.nstep + 1;
                 this.simTime =  (this.nstep-1)*this.dt;
                 double[] pos = measurements.getPosition();
                 double[] vel = measurements.getVelocity();
                 double[] accel = measurements.getAcceleration();



                 // write the measurements if enough time has passed
                 if (this.writeOutputFile == 1)
                 {
                     this.ElapsedTimeW = this.simTime - this.lastWriteTime;
                     if ( this.ElapsedTimeW>this.dTwrite )
                     {
                     try
                         {

                       this.out.write("" + simTime );
                       if (this.writePos == 1)
                       {
```

```
                this.out.write(" " + pos[0] + " " + pos[1] + " " + pos[2]
                        + " " + pos[3] + " " + pos[4] + " " + pos[5] );
                this.out.write(" " +  measPitch[0] + " " + measPitch[1] + " " + measPitch[2] );
                this.out.write(" " +  measBMZ[0] + " " + measBMZ[1] + " " + measBMZ[2] );
                this.out.write(" " +  measBMY[0] + " " + measBMY[1] + " " + measBMY[2] );
                this.out.write(" " +  oldAzi[0] + " " + oldAzi[1] + " " + oldAzi[2] );
                this.out.write(" " +  this.oldMy2cm + " " + this.oldMy3cm );
                }
                if (this.writeVel == 1)
                {
                this.out.write(" " +  vel[0] + " " + vel[1] + " " + vel[2]
                        + " " + vel[3] + " " + vel[4] + " " + vel[5]  );
                }
                if (this.writeAcc == 1)
                {
                this.out.write(" " +  accel[0] + " " + accel[1] + " " + accel[2]
                        + " " + accel[3] + " " + accel[4] + " " + accel[5]  );
                }

                this.out.newLine();
                this.out.flush();

                   }
                   catch (Exception e){//Catch exception if any
                System.err.println("Error: " + e.getMessage());
                }
              this.lastWriteTime = this.simTime;
               }
          }



          // send some results to Riflex
          feedback.setGearShaftOmega(this.newOmegaGf); // fixed in V3.7.23
          feedback.setGeneratedPower(measurements.getOmega()*this.newTorque/1000*this.Ng); // kW

          // only update the control actions if enough time has passed
          this.ElapsedTime = this.simTime - this.lastControlTime;

          for (int i = 0; i < nblades; i++)
          {
              this.measPsi[i] = this.oldAzi[i] + measurements.getOmega()*this.Steptime;
              if ( this.measPsi[i]>=Math.PI )
              {
                  this.measPsi[i] = this.measPsi[i] - 2*Math.PI;
              }

          }

          if ( this.dTcontroller<=this.ElapsedTime*1.00001 )
          {
              normalControl(omega_g,vel[0],BMY,BMZ); // call normal control routine (modificationCF)
              this.lastControlTime = this.simTime;
          }
           // send the commands to Riflex
          for (int i = 0; i < nblades; i++)
          {
              feedback.setPitchAngleReference(i, this.newPitchC[i]);
          }
          feedback.setTorqueReference( this.newTorque/1000*this.Ng);


          // Options other than normal control
          // warm-up with fixed initial pitch
         if (this.simTime < this.warmUpTime)
        {
            double WUpitch = 0;
            if(this.simTime < this.warmUpTime/2)
            {
            WUpitch = this.initialPitch*this.simTime/(this.warmUpTime/2);
            }

            else
            {
            WUpitch = this.initialPitch;
            }

            WUpitch = Math.min(Math.max(WUpitch,this.th_min),this.th_max);
            this.dPitch = Math.min( Math.max((WUpitch-this.measPitch[0])/this.ElapsedTime,this.dthmin)
              ,this.dthmax);
            WUpitch = this.measPitch[0] + this.dPitch*this.ElapsedTime;
            WUpitch = Math.min(Math.max(WUpitch,this.th_min),this.th_max);

             for (int j = 0; j < nblades; j++)
                  {
                  feedback.setPitchAngleReference(j,WUpitch);
                  this.oldPitchC[j] = WUpitch;
                  this.newPitchC[j] = WUpitch;
                  }
         }
```

```
                else
                { // fixed pitch
                    if (this.fixedPitchOn == 1)
                    {
                      for (int j = 0; j < nblades; j++)
                          {
                          feedback.setPitchAngleReference(j,this.fixedPitch);
                          this.oldPitchC[j] = this.fixedPitch;
                          this.newPitchC[j] = this.fixedPitch;
                          this.dPitch = 0.0;
                          }

                    }



                }


        }

/*************************************************************************/
/* Normal control routine for pitch*/
        private void normalControl(double omega_g, double SurgeVel, double[] BMY, double[] BMZ) //modificationCF
        {
                    // filter the generator speed
            double Alpha = Math.exp(-1*this.ElapsedTime*this.cornerFreq );
            this.newOmegaGf = (1.0-Alpha)*omega_g +
                    Alpha*this.oldOmegaGf;
            double Alpha2 = Math.exp(-1*this.ElapsedTime*this.cutoffFreq);
            this.newSurgeVel = (1.0-Alpha2)*SurgeVel +
                    Alpha2*this.oldSurgeVel;
            //this.newSurgeVel = SurgeVel;

            double deltaOmegaGf = this.newOmegaGf - this.omega_g_nom;
            double deltaSurgeVel = this.newSurgeVel - 0;
            // Computer other torque control parameters
            // modificationCF
            //double omega_g_var = this.omega_g_nom - this.dTcontroller*deltaSurgeVel*1346000/100 -
                    //(this.dTcontroller-100)/100*deltaOmegaGf;
            double VS_RtGnSp = 0.99*this.omega_g_var;
            // double VS_RtGnSp = 0.99*this.omega_g_nom;
            double VS_SySp = VS_RtGnSp/(1+0.01*this.VS_SlPc);
            double VS_Slope15=(this.VS_Rgn2K*this.VS_Rgn2Sp*this.VS_Rgn2Sp)/
                    (this.VS_Rgn2Sp - this.VS_CtInSp);
            double VS_Slope25 = (this.Pnom/VS_RtGnSp)/(VS_RtGnSp-VS_SySp);
            double VS_TrGnSp = 0;
            if (this.VS_Rgn2K == 0)
            {
                VS_TrGnSp = VS_SySp;
            }
            else
            {
                VS_TrGnSp = (VS_Slope25 - Math.sqrt(VS_Slope25*
                (VS_Slope25-4.0*this.VS_Rgn2K*VS_SySp)))/
                        (2*this.VS_Rgn2K);
            }

/*************************************************************************/
            //double ratio = this.omega_g_nom/this.omega_g_var;

                if ((this.newOmegaGf >= VS_RtGnSp) ||
                   ((this.simTime>=this.warmUpTime) & (Math.abs(this.oldPitchC[0]) >= this.VS_Rgn3MP) & this.
                        fixedPitchOn == 0))
            { // region 3
                // collective pitch (energy shaping)
                this.deltaomega_g_var = - this.dTcontroller/0.048*deltaSurgeVel*62 - (this.dTcontroller-0.048)/0.048*
                       deltaOmegaGf;
                this.omega_g_var = this.omega_g_nom + this.deltaomega_g_var;

                // individual pitch
                //double My2cm = (2*Math.sin(this.measPsi[0])*this.measBMY[0] + 2*Math.sin(this.measPsi[1])*this.
                       measBMY[1]
                //        + 2*Math.sin(this.measPsi[2])*this.measBMY[2])/3;
                //double My3cm = (2*Math.cos(this.measPsi[0])*this.measBMY[0] + 2*Math.cos(this.measPsi[1])*this.
                       measBMY[1]
                //        + 2*Math.cos(this.measPsi[2])*this.measBMY[2])/3;

                // Filtering blade root moment
                double Alpha3 = Math.exp(-1*this.ElapsedTime*this.halfFreq*2);

                this.measBMY[0] = (1.0-Alpha3)*BMY[0] +
                    Alpha3*this.oldmeasBMY[0];
                this.measBMY[1] = (1.0-Alpha3)*BMY[1] +
                    Alpha3*this.oldmeasBMY[1];
                this.measBMY[2] = (1.0-Alpha3)*BMY[2] +
                    Alpha3*this.oldmeasBMY[2];

                //this.measBMZ[0] = (1.0-Alpha3)*BMZ[0] +
```

```
1499                //    Alpha3*this.oldmeasBMZ[0];
1500                //this.measBMZ[1] = (1.0-Alpha3)*BMZ[1] +
1501                //    Alpha3*this.oldmeasBMZ[1];
1502                //this.measBMZ[2] = (1.0-Alpha3)*BMZ[2] +
1503                //    Alpha3*this.oldmeasBMZ[2];
1504
1505                //this.My2cm = (this.measBMY[0]*Math.cos(this.measPitch[0]+this.twist) - this.measBMZ[0]*Math.sin(this.
                          measPitch[0]+this.twist))*Math.cos(this.oldAzi[0])
1506                //          + (this.measBMY[1]*Math.cos(this.measPitch[1]+this.twist) - this.measBMZ[1]*Math.sin(this.
                          measPitch[1]+this.twist))*Math.cos(this.oldAzi[1])
1507                //          + (this.measBMY[2]*Math.cos(this.measPitch[2]+this.twist) - this.measBMZ[2]*Math.sin(this.
                          measPitch[2]+this.twist))*Math.cos(this.oldAzi[2]);
1508
1509                //this.My3cm = (this.measBMY[0]*Math.cos(this.measPitch[0]+this.twist) - this.measBMZ[0]*Math.sin(this.
                          measPitch[0]+this.twist))*Math.sin(this.oldAzi[0])
1510                //          + (this.measBMY[1]*Math.cos(this.measPitch[1]+this.twist) - this.measBMZ[1]*Math.sin(this.
                          measPitch[1]+this.twist))*Math.sin(this.oldAzi[1])
1511                //          + (this.measBMY[2]*Math.cos(this.measPitch[2]+this.twist) - this.measBMZ[2]*Math.sin(this.
                          measPitch[2]+this.twist))*Math.sin(this.oldAzi[2]);
1512
1513                this.My2cm = 2*this.measBMY[0]*Math.sin(this.oldAzi[0])/3
1514                           + 2*this.measBMY[1]*Math.sin(this.oldAzi[1])/3
1515                           + 2*this.measBMY[2]*Math.sin(this.oldAzi[2])/3;
1516
1517                this.My3cm = 2*this.measBMY[0]*Math.cos(this.oldAzi[0])/3
1518                           + 2*this.measBMY[1]*Math.cos(this.oldAzi[1])/3
1519                           + 2*this.measBMY[2]*Math.cos(this.oldAzi[2])/3;
1520
1521                if (this.ConstPower == 1)
1522                {  // constant power
1523                this.newTorque = this.Pnom/this.newOmegaGf;
1524                }
1525                else
1526                { // constant torque
1527                this.newTorque = this.Tnom;
1528                }
1529            }
1530            else if (this.newOmegaGf <= this.VS_CtInSp)
1531            { // region 1
1532                this.newTorque = 0.0;
1533            }
1534            else if (this.newOmegaGf < this.VS_Rgn2Sp)
1535            { // region 1.5
1536                this.newTorque = VS_Slope15*
1537                         (this.newOmegaGf - this.VS_CtInSp);
1538            }
1539            else if (this.newOmegaGf < VS_TrGnSp)
1540            { // region 2
1541                this.newTorque = this.VS_Rgn2K*
1542                         Math.pow(this.newOmegaGf,2);
1543            }
1544            else
1545            { // region 2.5
1546                this.newTorque = VS_Slope25*(this.newOmegaGf - VS_SySp);
1547            }
1548
1549 /*****************************************************************************/
1550            // PI controller for pitch angle
1551
1552            // Speed error
1553            // modificationCF
1554            double Ep = this.newOmegaGf - omega_g_var;
1555            //double Ep = this.newOmegaGf - this.omega_g_nom;
1556            this.newPerr = Ep;
1557
1558            // integrated speed error
1559            double Ei = this.oldIntErr + Ep*this.ElapsedTime;
1560
1561
1562             // gain scheduling correction factor
1563            double GK = 1.0/(1.0 + this.oldPitchC[0]/this.kK ); //note pitch1 is in rad!
1564
1565
1566            if ((this.endWarmUp == 0) & (this.simTime >= this.warmUpTime))
1567            {
1568                Ei = this.initialPitch/this.kI/GK;
1569                this.oldIntErr = Ei;
1570                this.newIntErr = Ei;
1571                System.out.println("End of controller warm-up");
1572                this.endWarmUp = 1;
1573            }
1574
1575
1576            // saturate integral term
1577            Ei = Math.min(Math.max(Ei,this.th_min/(GK*this.kI)),this.th_max/(GK*this.kI));
1578
1579
1580           // compute pitch commands
1581            double PitComP = GK*this.kP*Ep;
1582            double PitComI = GK*this.kI*Ei;
```

```
1583
1584
1585            double PitComT = PitComP + PitComI;
1586            this.newIntErr = Ei;
1587
1588  /*************************************************************************/
1589            // saturation limits of control signals
1590            this.newTorque =
1591                  Math.min(Math.max(this.newTorque,this.Qgmin),this.Qgmax);
1592            double dTorque =
1593                   Math.min(
1594                   Math.max((this.newTorque-this.oldTorque)/this.ElapsedTime,this.dQgmin)
1595                   ,this.dQgmax);
1596            this.newTorque = this.oldTorque + dTorque*this.ElapsedTime;
1597
1598            double GS = 1/(1+4.55*PitComT);
1599
1600            double EiMy2 = this.oldIntMy2cm + this.My2cm*this.ElapsedTime;
1601            double EiMy3 = this.oldIntMy3cm + this.My3cm*this.ElapsedTime;
1602
1603            // saturation if integral term
1604            EiMy2 = Math.min(Math.max(EiMy2,this.th_min/(GS*this.kI_ipc)),this.th_max/(GS*this.kI_ipc));
1605            EiMy3 = Math.min(Math.max(EiMy3,this.th_min/(GS*this.kI_ipc)),this.th_max/(GS*this.kI_ipc));
1606
1607            double th_y2cm = GS*kP_ipc*this.My2cm + GS*kI_ipc*EiMy2 + kD_ipc*(this.My2cm-this.oldMy2cm)/this.
                  ElapsedTime;
1608
1609            double th_y3cm = GS*kP_ipc*this.My3cm + GS*kI_ipc*EiMy3 + kD_ipc*(this.My3cm-this.oldMy3cm)/this.
                  ElapsedTime;
1610
1611
1612            this.ipcPitch[0] = PitComT + Math.sin(this.oldAzi[0])*th_y2cm + Math.cos(this.oldAzi[0])*th_y3cm;
1613            this.ipcPitch[1] = PitComT + Math.sin(this.oldAzi[1])*th_y2cm + Math.cos(this.oldAzi[1])*th_y3cm;
1614            this.ipcPitch[2] = PitComT + Math.sin(this.oldAzi[2])*th_y2cm + Math.cos(this.oldAzi[2])*th_y3cm;
1615
1616            // ipcPitch is used instead of PitComT
1617            for (int i = 0; i < nblades; i++)
1618            {
1619                this.ipcPitch[i] = Math.min(Math.max(this.ipcPitch[i],this.th_min),this.th_max);
1620                double ddes = (this.ipcPitch[i]-this.measPitch[i])/this.ElapsedTime;
1621                dPitch = Math.min(Math.max(ddes,this.dthmin),this.dthmax);
1622                this.newPitchC[i] = this.oldPitchC[i] + dPitch*this.ElapsedTime; //dPitch*this.dTcontroller;//
1623                this.oldPitchC[i] = this.newPitchC[i];
1624            }
1625
1626
1627
1628
1629
1630            // other regions
1631                //PitComT =
1632                //    Math.min(Math.max(PitComT,this.th_min),this.th_max);
1633                //for (int i = 0; i < nblades; i++)
1634                //    {
1635                //    double ddes = (PitComT-this.measPitch[i])/this.ElapsedTime;
1636                //    dPitch = Math.min(Math.max(ddes,this.dthmin),this.dthmax);
1637                //    this.newPitchC[i] = this.oldPitchC[i] + dPitch*this.ElapsedTime; //dPitch*this.dTcontroller;//
1638                //    this.oldPitchC[i] = this.newPitchC[i];
1639                //    }
1640
1641
1642
1643
1644            //PitComT =
1645            //      Math.min(Math.max(PitComT,this.th_min),this.th_max);
1646
1647
1648
1649
1650  /*************************************************************************/
1651            // set the variables for the next call
1652            this.oldMy2cm = this.My2cm;
1653            this.oldMy3cm = this.My3cm;
1654            this.oldIntMy2cm = EiMy2;
1655            this.oldIntMy3cm = EiMy3;
1656            this.oldmeasBMY = this.measBMY;
1657            this.oldmeasBMZ = this.measBMZ;
1658
1659            this.oldIntErr = this.newIntErr;
1660            this.oldTorque = this.newTorque;
1661            this.oldOmegaGf = this.newOmegaGf;
1662            this.oldSurgeVel = this.newSurgeVel;
1663            this.oldOmega_g_var = omega_g_var;
1664            this.oldAzi = this.measPsi;
1665
1666        }
1667
1668  /*************************************************************************/
1669  /*************************************************************************/
1670
```

```
    /**
     * Called once after simulation is done
     */
    @Override
    public void finish()
        {
            if (this.writeOutputFile == 1)
            {
         try
                {
                   this.out.flush();
                   this.out.close();
                }
             catch (Exception e)
                {//Catch exception if any
                   System.err.println("Error: " + e.getMessage());
                }
            }


    }

}
```

TurbSim input file

```
---------TurbSim v2.00.* Input File------------------------
Example input file for TurbSim.
---------Runtime Options-----------------------------------
False      Echo            - Echo input data to <RootName>.ech (flag)
-1337032771 RandSeed1        - First random seed  (-2147483648 to 2147483647)
-503679729      RandSeed2       - Second random seed (-2147483648 to 2147483647) for intrinsic pRNG, or an alternative
       pRNG: "RanLux" or "RNSNLW"
False      WrBHHTP         - Output hub-height turbulence parameters in binary form?  (Generates RootName.bin)
False      WrFHHTP         - Output hub-height turbulence parameters in formatted form?  (Generates RootName.dat)
False      WrADHH          - Output hub-height time-series data in AeroDyn form?  (Generates RootName.hh)
False      WrADFF          - Output full-field time-series data in TurbSim/AeroDyn form? (Generates RootName.bts)
True       WrBLFF          - Output full-field time-series data in BLADED/AeroDyn form?  (Generates RootName.wnd)
False      WrADTWR         - Output tower time-series data? (Generates RootName.twr)
False      WrFMTFF         - Output full-field time-series data in formatted (readable) form?  (Generates RootName.u,
       RootName.v, RootName.w)
False      WrACT           - Output coherent turbulence time steps in AeroDyn form? (Generates RootName.cts)
True       Clockwise       - Clockwise rotation looking downwind? (used only for full-field binary files - not necessary
        for AeroDyn)
 0         ScaleIEC        - Scale IEC turbulence models to exact target standard deviation? [0=no additional scaling;
       1=use hub scale uniformly; 2=use individual scales]

--------Turbine/Model Specifications-----------------------
32         NumGrid_Z       - Vertical grid-point matrix dimension
32         NumGrid_Y       - Horizontal grid-point matrix dimension
0.05       TimeStep        - Time step [seconds]
11000.0    AnalysisTime    - Length of analysis time series [seconds] (program will add time if necessary: AnalysisTime
       = MAX(AnalysisTime, UsableTime+GridWidth/MeanHHWS) )
11000.0    UsableTime      - Usable length of output time series [seconds] (program will add GridWidth/MeanHHWS seconds
       unless UsableTime is "ALL")
90.00      HubHt           - Hub height [m] (should be > 0.5*GridHeight)
160.00      GridHeight      - Grid height [m]
160.00      GridWidth       - Grid width [m] (should be >= 2*(RotorRadius+ShaftLength))
0          VFlowAng        - Vertical mean flow (uptilt) angle [degrees]
0          HFlowAng        - Horizontal mean flow (skew) angle [degrees]

--------Meteorological Boundary Conditions-------------------
"IECKAI"   TurbModel    a     - Turbulence model ("IECKAI","IECVKM","GP_LLJ","NWTCUP","SMOOTH","WF_UPW","WF_07D","WF_14D
       ","TIDAL","API","USRINP","TIMESR", or "NONE")
"unused"   UserFile         - Name of the file that contains inputs for user-defined spectra or time series inputs (used
       only for "USRINP" and "TIMESR" models)
"3"        IECstandard      - Number of IEC 61400-x standard (x=1,2, or 3 with optional 61400-1 edition number (i.e. "1-
       Ed2") )
"C"        IECturbc         - IEC turbulence characteristic ("A", "B", "C" or the turbulence intensity in percent) ("
       KHTEST" option with NWTCUP model, not used for other models)
"NTM"      IEC_WindType     - IEC turbulence type ("NTM"=normal, "xETM"=extreme turbulence, "xEWM1"=extreme 1-year wind,
       "xEWM50"=extreme 50-year wind, where x=wind turbine class 1, 2, or 3)
default    ETMc             - IEC Extreme Turbulence Model "c" parameter [m/s]
"PL"       WindProfileType  - Velocity profile type ("LOG";"PL"=power law;"JET";"H2L"=Log law for TIDAL model;"API";"USR
       ";"TS";"IEC"=PL on rotor disk, LOG elsewhere; or "default")
"unused"   ProfileFile      - Name of the file that contains input profiles for WindProfileType="USR" and/or TurbModel="
       USRVKM" [-]
90.00    RefHt            - Height of the reference velocity (URef) [m]
12.0       URef             - Mean (total) velocity at the reference height [m/s] (or "default" for JET velocity profile)
        [must be 1-hr mean for API model; otherwise is the mean over AnalysisTime seconds]
350    ZJetMax          - Jet height [m] (used only for JET velocity profile, valid 70-490 m)
0.14    PLExp            - Power law exponent [-] (or "default")
0.0003    Z0               - Surface roughness length [m] (or "default")

--------Non-IEC Meteorological Boundary Conditions------------
```

```
default   Latitude        - Site latitude [degrees] (or "default")
0.05      RICH_NO         - Gradient Richardson number [-]
default   UStar           - Friction or shear velocity [m/s] (or "default")
default   ZI              - Mixing layer depth [m] (or "default")
default   PC_UW           - Hub mean u'w' Reynolds stress [m^2/s^2] (or "default" or "none")
default   PC_UV           - Hub mean u'v' Reynolds stress [m^2/s^2] (or "default" or "none")
default   PC_VW           - Hub mean v'w' Reynolds stress [m^2/s^2] (or "default" or "none")

--------Spatial Coherence Parameters----------------------------
default   SCMod1           - u-component coherence model ("GENERAL", "IEC", "API", "NONE", or "default")
default   SCMod2           - v-component coherence model ("GENERAL", "IEC", "NONE", or "default")
default   SCMod3           - w-component coherence model ("GENERAL", "IEC", "NONE", or "default")
default   InCDec1          - u-component coherence parameters for general or IEC models [-, m^-1] (e.g. "10.0  0.3e-3"
       in quotes) (or "default")
default   InCDec2          - v-component coherence parameters for general or IEC models [-, m^-1] (e.g. "10.0  0.3e-3"
       in quotes) (or "default")
default   InCDec3          - w-component coherence parameters for general or IEC models [-, m^-1] (e.g. "10.0  0.3e-3"
       in quotes) (or "default")
default   CohExp           - Coherence exponent for general model [-] (or "default")

--------Coherent Turbulence Scaling Parameters-------------------
".\EventData"  CTEventPath     - Name of the path where event data files are located
"Random"  CTEventFile     - Type of event files ("LES", "DNS", or "RANDOM")
true      Randomize       - Randomize the disturbance scale and locations? (true/false)
 1.0      DistScl         - Disturbance scale [-] (ratio of event dataset height to rotor disk). (Ignored when
        Randomize = true.)
 0.5      CTLy            - Fractional location of tower centerline from right [-] (looking downwind) to left side of
        the dataset. (Ignored when Randomize = true.)
 0.5      CTLz            - Fractional location of hub height from the bottom of the dataset. [-] (Ignored when
        Randomize = true.)
30.0      CTStartTime     - Minimum start time for coherent structures in RootName.cts [seconds]

====================================================
! NOTE: Do not add or remove any lines in this file!
====================================================
```